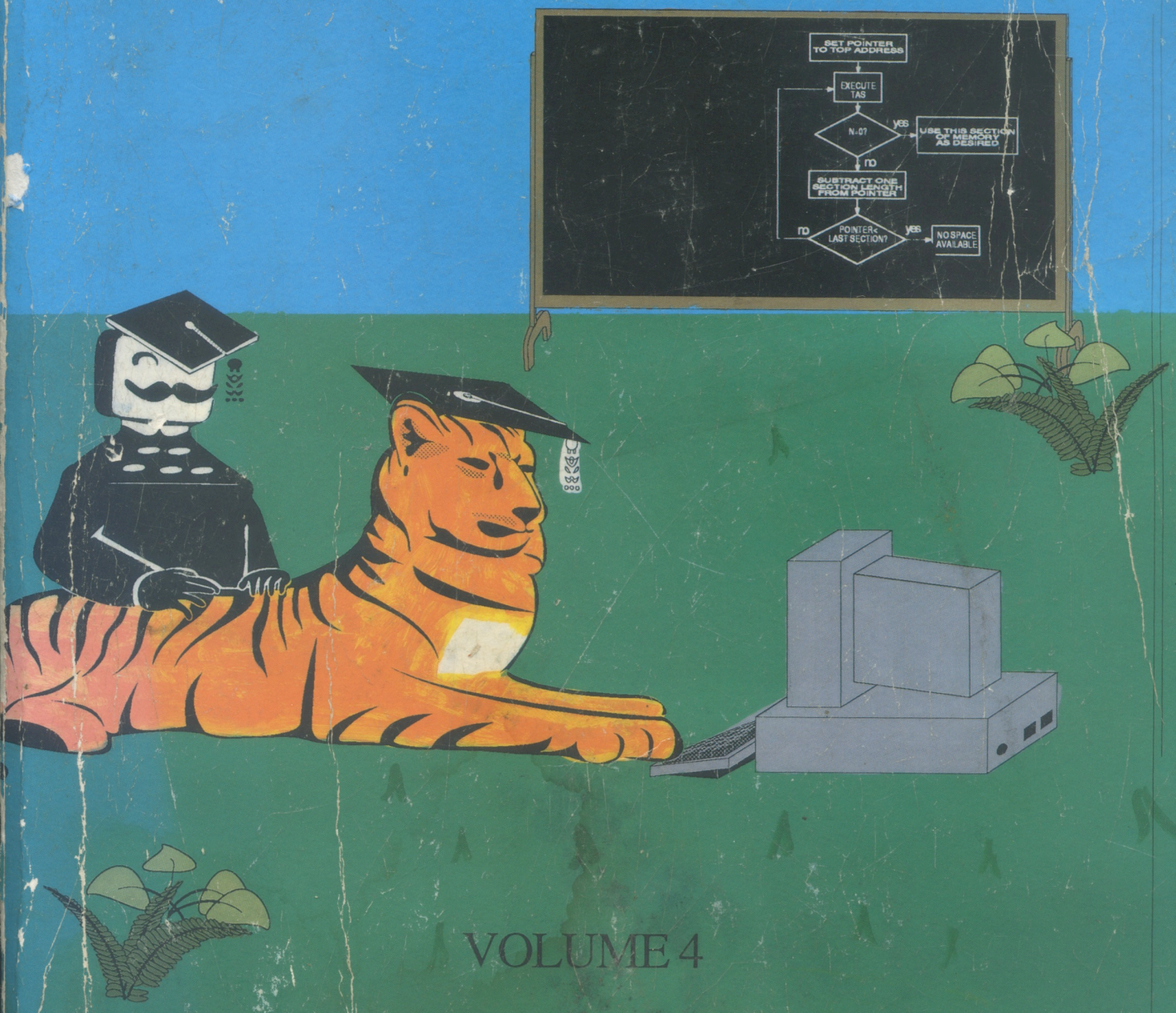
The Definitive

MacTutor

The Macintosh Programming Journal



VOLUME 4

The Definitive MacTutor

The Macintosh Programming Journal

Volume 4

Edited by
David E. Smith

Published by
MacTutor®

© The Definitive MacTutor, Vol. 4

© 1989 by MacTutor®, Anaheim CA. 92807

This book was created on a Macintosh Plus Computer
using a LaserWriter Plus and Aldus Pagemaker Software.

All rights reserved. No part of this book
may be reproduced, in any form or by any means,
without permission in writing from the publisher.

Printed in the United States of America
by Webcraft Craftmanship Printing, Garden Grove, CA 92641

10 9 8 7 6 5 4 3 2

ISBN 0-9617544-3-5

MacTutor® and the MacTutor Man Logo are registered trademarks of the MacTutor Company.

MacTutor®
The Macintosh Programming Journal
P.O. Box 400
Placentia, CA 92670
or
1240 Van Buren, #105
Anaheim, CA 92807

I would like to thank Kirk Chase for preparing this manuscript for publication and Mike Gibson for his fine artwork. Also I would like to thank the contributing editors and all who contributed this past year to MacTutor.

Finally, I would like to thank my wife, Laura, for her support and all the office staff at MacTutor. They are a pleasure to work with.

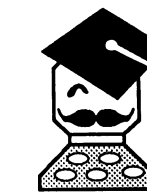
Table Of Contents

The Definitive MacTutor, Volume 4, 1988

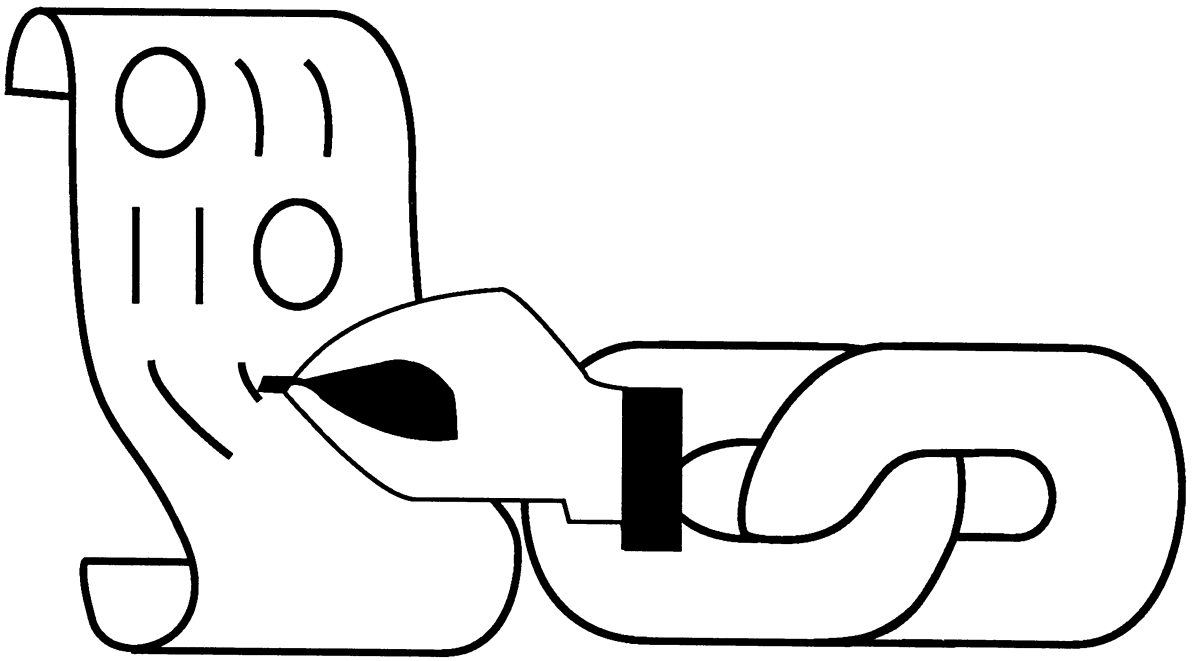
Title	(Source Disk)	Page
Table of Contents		v...vi
Assembly		1...75
Modifying Apple's SCSI Driver	(28)	2
Looking At Text From a Different Angle	(29)	9
QuickTrap Routines Bypass Trap Dispatcher	(30)	21
FKEY that runs other FKEYs!	(33)	25
Font Dialog Box Using List Manager	(34)	31
Faster BitMap Rotation	(38)	56
Smoothing via PicComments	(39)	61
C		76...254
Breaking the Four Window Barrier	(28)	77
Overview & Downloader in C	(28)	79
Smart Documents	(29)	93
A Background Task for Measuring Load	(30)	100
Tear-off Menus & Floating Palettes	(31)	112
A Vaccine for the 'nVIR' Virus	(32)	150
Protocols for IAC Driver, Part I		155
IAC Driver, Part II	(37)	162
MenuStuff for Handling Menus	(37)	177
HFS Transfer DA	(37)	189
IAC Driver Demo in MPW C	(38)	201
Tool Window Manager	(39)	230
Pascal		255...406
Keyboard Wars!	(28)	256
Format Drivers Bring Freedom of Revision	(28)	264
MultiFinder Friendly MacDraw Plotter	(29)	274
Screen Dump FKEY for Color Picts	(29)	295
Graphic Devices & Chroma	(30)	303
The Palette Manager & Color Puzzle	(31)	314
Palette Animation	(32)	324
Serial Port Demo	(33)	338
Menus as Objects in TML Pascal	(33)	349
4th Dimension® Externals	(36)	356
Pop-up Menus CDEF	(36)	364
Fun With Regions, Part II	(36)	380
Menus In Windows	(38)	388
Inside Macinkeys	(39)	396
Basic		407...454
Color & DragGrayRgn Explored	(28)	408
Updating Windows in Basic	(30)	412
Color Mixer Creates Custom Colors	(31)	415
Quick & Dirty Data Conversion	(32)	419
Hierarchical Menus & Color Notes	(33)	422
True Basic Playes the Simon Game	(34)	428
True Basic & Prototyper Reviewed	(35)	432
Matrices in True Basic	(36)	442
Basic Wars and True Windows	(37)	446
System Snooping in MS QuickBasic	(38)	449
QuickBasic's Statement Mover	(39)	452
Forth		455...533
A SCSI Driver in Forth	(29)	456
Time Manager Reminder DA	(30)	464
Direct 68881 Floating Point Access	(31)	469
Formatted Text with Style	(32)	474
Graf3D Library Access	(33)	481
Objective Activities in Forth	(34)	487
Object Forth Project	(35)	497
AppleTalk Mail	(36)	509
The Notification Manager	(37)	517
Code Segments & Linker	(38)	525
European Developers' Conference	(39)	531
Fortran		534...550
Tips for Engineering Applications	(29)	535
MatLab Comes to the Mac II	(32)	539
Fortran Printing Interface	(35)	543
The Status of Fortran		547
Lisp		551...567
MacScheme vs. Coral Lisp		552
Lisp Objects: Ford vs. Chevy		565
HyperChat		568...629
New HyperCard Products	(32)	569
PopUpMenus in HyperTalk	(32)	573
New HyperCard 1.2 Features	(33)	580
XCMD CookBook	(33)	584
HyperTools for HyperProgrammers	(34)	588
XCMD Corner	(34)	593
AutoHyperEdit Moves in Groups	(35)	596
XCMD Cookbook: Sorting Routines	(35)	602
ScriptTips	(36)	607
Comparing HyperTalk to Pascal	(36)	609
XCMD Corner: Importing Text	(36)	615
Happy Birthday		619
XCMD Cookbook: Print Manager	(37)	620
XCMD Cookbook: Assembly Interface	(38)	624
The Vertical Market	(39)	627
More Articles		630...685
Using Consulair C & TML in MPW	(28)	631

© The Definitive MacTutor, Vol. 4

Title	(Source Disk)	Page
Ask Apple Technical Support	(30)	648
Calculating Distance for Navigation	(30)	652
MacBox and the Serial Sermon		660
Comments About PICTs		662
System Sleuthing II: The Sequel	(34)	669
Prototyper, MacExpress, and FaceIt Compared		675
OS/2 Presentation Manager		681
Letters		**686...713**
Mousehole		**714...745**
Topical Index		**747...750**
Author's Index		**752...753**



© The Definitive MacTutor, Vol. 4



Assembly Lab

© The Definitive MacTutor, Vol. 4

Advanced Mac'ing

Modifying Apple's SCSI Driver

Jörg Langowski
MacTutor Editorial Board
Grenoble, France
MacTutor Vol. 4 No. 1

How to Make Apple's 'non-functional' SCSI Driver, Functional!

Happy New Year and welcome to our fourth volume. We'll step right in the middle of things with a SCSI driver project that will take us two columns.

You've seen several articles on SCSI device handling in MacTutor; Tim Standing's contributions in V3#2 and V3#6, and my own column in V3#3. We showed you how to hook up the electronics of a generic SCSI disk to the Mac, how to play with SCSI commands and do raw data transfers, and how to format the SCSI disk. One crucial thing has always been missing: the SCSI driver itself.

History

It's been over a year now that I hooked up my own 80 MByte disk, a Quantum Q280, to my Mac (Plus at that time), and was confronted with all those problems, including getting a driver that would make my disk work. That last part turned out to be the hard one, at that time.

All information that I could then get hold of was the source of a 'generic' SCSI driver published by Apple in a Software Supplement, and also available on various bulletin boards. The problem with this driver was that it wouldn't quite work as promised. In this column I'll explain the modifications that I had to do to the Apple driver to make it boot and run *on the Quantum Q280 disk.* I am not claiming that it will drive any SCSI disk, although in theory it should drive many of them; in practice, however, a Seagate ST225N will run with the *original* code supplied by Apple, but not with my code. However, by describing my implementation and its differences to the original, I hope to give you some hints what can be done to make an arbitrary SCSI disk work with a fully or partially home-brew driver.

Some of you might find this information out of date, since by now the HD SC installer has been released by Apple. Alas: I lost a lot of time backing up some 63 Megabytes to run the HD SC installer on my Q280; only to find out that the drive wasn't recognized and I had to re-install my old driver. Which, as you'll see soon, is admittedly some kind of a kludge, but has performed without problems for over one year now.

It is for exactly the reason that, after all, SCSI devices are not *that* generic, that David Smith asked me to write about my SCSI driver [Apple Expo, Paris Sept 30: "You really have a SCUZZY driver? Well, publish it!"]. Therefore, not much Forth this month. Rather assembler, and general ramblings. Next month I'll continue with a re-implementation of Apple's SCSI driver in ... Forth. Yes, that's right. Just to get all you assembly/ hardware guys out there to buy your copies of Mach2.

Apple's 'generic' SCSI driver

There are two problems with printing the full, commented source listing of Apple's driver here (I already mentioned this in the March 87 article). First, although it is almost two years old, it is copyrighted by Apple. Second, the source is quite long, (some 12 pages with comments) and it is available through APDA, Delphi, Compuserve and who knows where. I think it is much simpler that you download a copy of the [heavily commented] driver's source - I got mine from Delphi - and follow the explanations in this article to understand what it's doing. The parts of the driver that I modified are printed in the listing. The complete source of a SCSI driver, written in Forth, will follow next month.

[The SCSI driver example is a product of APDA, part # KMSSDP, which sells for $10. The APDA catalog says, and I quote: 'Contains a heavily commented, but non-functional, sample scsi driver written in Assembly language'. Jörg's article should help make it functional! We can't distribute it since APDA is selling it as a product, so please contact them for this source. You'll also want to make sure you have the 2.0b1 MPW Shell release, which contains the latest assembler equates including the SCSI equates. Included with the scsi example is a eight page handout which says: 'It is provided to shows how such a driver might be written (by someone with not a lot of time to tell developers how to write drivers!)'. And again, this little gem: 'Unfortunately, if you happen to have gotten a good deal on an old 370 meg drive that you want to hook up to your Macintosh, we can't really help you (there are only six of us for all Macintosh development, and four thousand of you!)'. The comments go on to state that the Technical Support Group's charter is to help developer's in creating products for market, so I assume any subject that does not appear to lead to product development is given little developer support! I have always disagreed with this narrow view of software development. The amount of Macintosh software developed in-house far exceeds what little commercial "MS Word" type program development that is going on, and this will only increase as the Mac II moves into big-time engineering departments. Computers were tools for problem solving long before they were a means to help Apple sell more computers. -Ed]

The structure of a SCSI driver

Like any driver, the SCSI driver will have a header before the actual driver code. This header looks like the following:

```
Flags   [word]    $4F00
        ;read, write, control, status, needs lock
```


© The Definitive MacTutor, Vol. 4

```
        [word]   $0   ;no periodic action
        [word]   $0   ;no event mask
        [word]   $0   ;no menu

; Entry point offset table
        [word]    offset to Open routine
        [word]    offset to Prime routine
        [word]    offset to Control routine
        [word]    offset to Status routine
        [word]    offset to Close routine
        [counted string] the driver name
```

The block number at which the driver can be found on the disk, and its length, are given in the device descriptor map (DDM) on block 0 of the SCSI disk. On bootup, the driver is found (hopefully) at the location given in the DDM, read from the disk, and a jump is made to the beginning of the driver. Therefore, an SCSI driver must contain executable code *before* the header. Usually, a JMP or BRA instruction is contained here that jumps to a piece of code which does the actual initialization and installation.

SCSI driver installation

The procedure by which the Macintosh will install the driver on bootup is partly described in IM IV-293, partly in the comments to Apple's SCSI driver source. I'll summarize the relevant information here.

On bootup, the Macintosh will try to select a device on the SCSI bus by its ID, starting with ID=6 and going downward. In the older systems, the highest ID found was used as the number of the startup device; in the new system releases, the control panel allows you to select where you want to boot from.

The driver descriptor and device partition maps are then read from block 0 and 1 of the startup device. Their signatures are checked; the first word of block 0 must be $4552, and of block 1, $5453. The driver descriptor map (DDM) must contain an entry that corresponds to a driver for the Macintosh (processor type = 1, as explained in V3#3). The driver is read from the blocks given in the DDM and put into memory allocated in the system heap. Then the Macintosh jumps to the beginning of the driver code.

The driver code is supposed to 'install' itself. In Apple's code, the installation routine is located at the very end of the driver. It takes three parameters on entry:

- A0, a pointer to the device partition map
- D5, the SCSI ID of the device for this driver
- D7, default data start area', not used in the Apple driver.

I'll explain the steps the installer goes through. First, we have to let the system know that we exist. The table of existing drivers is the *unit table* (IM II-191), a pointer to which is available through the system global *UTableBase* ($11C). The unit table contains a handle to the unit's device control entry at *UTableBase+unitnumber*4*. By definition, the unit number of the SCSI device is its SCSI ID + 32. The driver reference number for a device is the two's complement of its unit number.

We therefore know our unit number and reference number when the installation routine is entered. The trap *DrvrInstall* (Toolbox $A03D, and not documented in IM), called with the driver reference number in D0, will allocate a new device control entry for the driver and put a handle to this DCE into the unit table. The format of a DCE, as given in IM II-190, looks like:

```
TYPE DCtlEntry = RECORD
      DCtlDriver : Ptr ;
           (ptr to ROM or handle to RAM driver)
      DCtlFlags : Integer ; (flags)
      DCtlQHdr : QHdr ; (driver's i/o queue)
      DCtlPosition : LongInt ;
           (byte pos used by read and write calls)
      DCtlStorage : Handle ;
           (hndl to RAM drivers private storage)
      DCtlRefNum : Integer ;
           (driver's reference number)
      DCtlCurTicks : LongInt ;
        (long counter for timing system task calls)
      DCtlWindow : Ptr ;
           (ptr to driver's window if any)
      DCtlDelay : Integer ;
           (number of ticks btwn sysTask calls)
      DCtlEMask : Integer ;
           (desk acessory event mask)
      DCtlMenu : Integer ;
           (menu ID of menu associated with driver)
      END ;
```

After the DCE has been created, some of its fields (`DCtlDriver, DCtlFlags, DCtlDelay, DCtlEmask, and DCtlMenu`) are filled using the information in the driver header. The driver will be marked "ROM based" (bit 6 of *dCtlFlags* = 0), because the header contains a pointer to the driver.

Next, the *Open* trap is called with the driver's name as its parameter. If the driver cannot be opened for some reason, *DrvrRemove* (Toolbox $A03E, also undocumented) is called with the reference number in D0, and the memory allocated to the driver is disposed with a call to *DisposPtr*.

If the driver has been opened correctly, the device partition map is checked for a partition with the file system ID 'TFS1'. The offset to this partition and its size are remembered in the local variables area of the driver. It is this partition which will be mounted as a Macintosh volume.

Finally, if the install routine has been called at boot time, the *dNeedTime* flag is set in the *DCtlFlags* word; this way the driver gets called by the desk manager when the system comes up. This will allow us to post a disk-insert event for this volume at that time. The reason to do this is that on boot-up the system remembers only the drive that was actually used to boot from. Thus, in case the boot was made from another drive (such as a floppy), our SCSI drive will have to be re-mounted, even though the driver has already been installed.

The Open routine

The Open routine will get memory for the driver's local variables and add the SCSI drive to the drive queue. Entry parameters to this routine are, as for all five driver routines (*Open, Control, Prime, Status, Close*) a pointer to a parameter block in A0 and to the device control entry in A1.

First, a block is reserved in the system heap for the local variables and its pointer stored in the driver's DCE. The drive number is initially set to 5, and the drive queue is traversed, starting at its head (contained in the system global *DrvQHdr+QHead* = $30A), to see whether the drive number is already in use. If it is, it is incremented by 1 and the queue checked again until an unused drive number has been found. This drive number is stored in a local variable. The drive queue element (IM II-127) for our drive is also set up in the local variable space and some of its fields are preset (see TN36, which also describes a routine that adds a drive to the drive queue in a very similar way):

The *qType* field is set to 1. This indicates that the drive size may be greater than 32 MBytes and that one more 16-bit word is following the *dQDriveSize* field of the drive queue element; this word indicates the high-order word of a 32-bit block count. With the old definition of the drive size (*qType* = 0), a 16-bit block count was used, allowing a maximum size of only 32 MBytes. We therefore have to remember that the high- and low-order words of the drive size are reversed when we are working with this variable later. The *dqDrvSize* is initialized to zero; this field will be filled by the *Install* routine after the driver is open. The *dQFSID* field is also set to 0 to indicate a normal HFS volume. The four bytes *preceding* the drive queue element contain flags (IM II-128); byte 1 of these flags is set to 8 to indicate a non-ejectable disk. The remaining flags are cleared.

The drive is added to the drive queue using the *AddDrive* trap ($A04E, not documented in IM) with D0 containing the drive number in its high order word and the driver reference number in its low order word, and A0 containing a pointer to the drive queue element.

The last action of the *Open* routine is to set up an SCSI pseudo program for read/write transfers in the local variable space. We use a very simple one:

```
SCNOINC   buffer, #bytes
SCSTOP
```

and this is where we first deviate from the Apple driver. Apple's example uses (and this works with the ST225N):

```
@1  SCINC  buffer, #bytes
    SCLOOP @1, count
    SCSTOP
```

which will transfer `count` blocks of `#bytes` each starting at address `buffer`. Our driver, on the contrary, will transfer all bytes requested in one single chunk. The reason for this will be explained later. First, keep yourself content with the fact that one method works on the Q280 and the other on the ST225N.

The Control routine

The SCSI driver must be able to respond to several control calls, distinguished by their *csCode* parameter; some of them are described in TN28. They are:

```
KillIo        csCode=1
Verify        csCode=5
Format        csCode=6
Eject         csCode=7
getIcon    csCode=21
AccRun        csCode=65
SCSICode      csCode=77
```

KillIo is ignored (the driver operates synchronously and won't return before the I/O operation is completed), as are *Verify* and *Format*.

Eject must be supported in some way. First, when the ROM code cannot boot off a volume, it will try to eject that volume; the driver code has to return with an error here, otherwise the Macintosh would keep trying to boot from this drive. Also, a disk inserted event is posted, so that later on , when the system comes up, the drive will be re-mounted.

getIcon, will return a pointer to an icon list (ICN#) corresponding to the disk icon, followed by a counted string identifying the driver.

AccRun will post a disk-inserted event (see above); it will be called by the desk manager when the system comes up after boot time. After being called, it will clear the*dNeedTime* flag in the DCE so that no further calls are made.

Finally, *SCSICode* is a control call that allows it to issue SCSI commands directly, with the parameters (SCSI command block pointer, its length, completion count, address of pseudo program) in the *CSParam* field of the parameter block.

Status and *Close* calls are not supported.

Read and Write

The actual read and write calls pass through the *Prime* routine. It is mainly this routine, and *SCSICommon* that handles SCSI commands, that I had to modify to make Apple's driver work with my Q280. The modified routines are printed in listing one.

Since the SCSI **Read** and **Write** commands only take an 8-bit sector count (512 bytes per sector), the author of Apple's driver decided to transfer the data in chunks of 64K maximum each. Furthermore, each such 'chunk' is transferred in 512 byte sub-chunks using the 'looping' SCSI pseudo program described above.

When I first tried to set up the Apple driver on the Q280 without any modifications, the routine 'almost' seemed to work, at least I could install my drive and see its icon on my desktop, although I couldn't boot. Nor could I read/write files larger than 64K. Desperate, I looked through Technical Note #96, which describes SCSI handling and known bugs.

The first problem was a well-known one with the Mac Plus ROM boot code: the Q280 was configured to enter the *Unit Attention* condition after a reset, and thus a read command right after that would fail. Although this problem could be overcome by changing the Unit Attention Enable bit on the drive with a *Mode Select* command, there was a second problem: The firmware that the Q280 used at that time took 4 seconds to recover after an SCSI bus reset, and since the Macintosh boot code issues resets in one-second intervals, the disk could never get ready for

the boot code to read the first blocks. This problem could only be solved by changing the firmware, which the local Quantum distributor eventually did.

But there were other problems, this time in the Apple SCSI driver. The multiple-chunk transfer instruction block seemed to be one of them. TN96 says: *"Don't use multi transfer TIBs. The SCSI Manager has trouble with losing bytes when executing more than one transferring TIB instruction within a single call. Try to code your TIB as simply as possible: SCNoInc followed by SCStop."*

Therefore, the *Open* routine was modified as described above. Although the problem was supposed to be fixed with System releases 3.2 or later, I found that a multi-transfer TIB wouldn't work even with more recent systems, and I still use the simple TIB shown above.

The second problem was that the original driver transferred the data from/to the disk by consecutive *Read* or *Write* commands if the amount of data exceeded 64K bytes. Even though each of these commands was issued in the proper way, *Get*ting the bus, *Select*ing the target, sending the command, doing the *SCSIRead* or *SCSIWrite*, and a *SCSIComplete* at the end, the driver seemed to be losing bytes when more than 64K of data were transferred. TN 96 cautions: *"Multiple calls to SCSIRead or SCSIRBlind after issuing a command and before calling SCSIComplete may not work."* Although the situation was not quite like this - each *Read* was in fact completed before the next one was started - I decided to transfer all bytes with one single SCSI command. The 10 byte commands *Read Extended* and *Write Extended* can be used for this purpose. They take a 16-bit sector count in their command block, and thus allow the transfer of 512*65536=32 Megabytes maximum. So until you've upgraded your machine to more than 32 Mbyte, this driver should have no problems.

Third, it seemed like blind transfers would not be accepted by my Q280. Therefore I changed the part of the *SCSICommon* routine that does the actual read/write so that only non-blind transfers would be made. This slows the driver down somewhat, but the overall performance (Disktimer II) is still about 60% of a Dataframe 40 XP on a Mac+.

Let me quickly summarize the main modifications that have to be made to Apple's driver to make it run with the Quantum Q280 SCSI disk:

- change the SCSI pseudo program so that only one transfer instruction is done.
- remove the main loop of the Prime routine that reads/writes the data in several pieces. Replace it by a sequence of instructions that uses the SCSI *Read/Write Extended* commands so that all data is transferred in one block.
- remove the option of doing blind reads. Only non-blind reads should be used.

The modified *Prime* and *SCSICommon* routines are printed in Listing 1. The code there makes reference to variables and routines of the Apple SCSI driver, so you should have its source at hand. The modified driver can be assembled with MPW or MDS; if you assemble it into a resource of type 'SDRV' and ID=128, the installer written in Mach2 (listing 2) will accept it and write it out to your SCSI drive. The installer code should be self-explanatory after you've read the installation strategy in V3#3.

Since this article has made reference to quite a few different sources, I'll summarize the recommended (or required) reading again:

- Apple example SCSI driver source, version 1.0 (APDA)
- Inside Macintosh:
  Vol. II, Device Manager; Vol. IV, SCSI Manager
- Technical Notes:
  28, Finders and Foreign Drives
  36, Drive Queue Element Format
  96, Known SCSI Bugs
- MacTutor:
  V3#2 p.67, Tim Standing, Build your own SCSI hard disk
  V3#3 p.32, Jörg Langowski, SCSI device introduction
  V3#6 p.45, Tim Standing, SCSI formatting program

The full source of a Forth SCSI driver is to follow in the next issue. See you then.

Listing 1: Modification of Apple's Prime and SCSICommon routines for the Quantum Q280

```
;
; Routine:    DiskPrime
; Arguments:  A0 (input)  - pointer to request parameter block
;             A1 (input)  - pointer to disk DCE
;
; The "prime" entry point is used for drive read/write.
;
; Within this routine, we use the following registers:
;   D0-D1: temps
;   D3: Next Sector Number
;   D5: Number of Physical Blocks Left to Xfer
;   D6: Number of blocks to transfer this iteration
;   A2: locals ptr
;   A3: paramblk ptr
;   A4: DCE Ptr

DiskPrime
    BSR.S   CkDrvNum  ;check for valid drive number,
                      ; setup A2 & D1
    SF   TickleFlag(A2)   ;Remember that we've been
                          ;accessed.

;Do the transfer. Our DCE ptr is in A4, our locals ptr in A2.
    MOVE.L  A0,A3 ;save the ioparamblk ptr in A3
                  ;Calculate sector count (to D5)
    MOVE.L IOByteCount(A3),D5 ;how many bytes does he want?
    BEQ.S   DiskParmErr     ;he doesn't want any! Ooops!
    LSR.L   #8,D5           ;convert bytes to 512-byte sectors
    LSR.L   #1,D5
@1  AND.L   #$001FFFFF,D5   ;make sure it fits in 21 bits
                ;Calculate starting sector number (to D3)
    MOVE.L  DCtlPosition(A4),D3     ;starting byte position
    LSR.L   #8,D3     ;convert sector # to 512-byte sectors
    LSR.L   #1,D3

    ;Range-check this operation:
    ;If (starting block number + block count) >= volume size
```

```
    ;then return IOErr.
    MOVE.L  D3,DØ  ;copy starting sector
    ADD.L   D5,DØ  ;add this request sector count
    CMP.L   D1,DØ  ;(block after end of request ?
                   ;  block after end of volume)
    BGE     PrimeErr

 ;This request is valid.
; Offset the starting sector # by the start of the partition
    ADD.L   Offset(A2),D3 ;offset to the partition start
                          ;Copy buffer address to NextAddr(A2)
    MOVE.L  IOBuffer(A3),NextAddr(A2)

;SCSI transfer is done by extended read or write.
;Therefore no loop, like in the original code, is necessary.
;The maximum # of bytes transferred this way
;will be 65535 sectors, i.e. 32 MBytes.
;This should be well above the buffer size of the Mac for
;some years to come. /JL
;
    MOVE.W  D5,D6  ;Transfer # of sectors requested.
        ; setup SCSI extended R/W block
        ;
       ;Stuff command byte first
@3  LEA     SCmd(A2),AØ     ;point at our command block
    MOVE.W  #$2800,DØ       ;assume Read command
    CMP.B   #aWrCmd,ioTrap+1(A3) ;are we writing?
    BNE.S   @4              ;  no.
    MOVE.W  #$2A00,DØ       ;  yes, do a Write command
@4  MOVE.W  DØ,(AØ)+        ;  put the command away.

     ;Stuff Starting sector number /JL
    MOVE.L  D3,(AØ)+  ;put it into SCSI command block
                      ; (clears logical unit number, too)
    ;Stuff block count, and clear the reserved byte.
    LSL.L   #8,D6     ;align D6 with cmd block structure
    MOVE.L  D6,(AØ)+  ;put it away.
    CLR.B   (AØ)+     ;clear the reserved byte.

     ;Set up the SCSI "pseudo-program". The SCSI manager
     ;will interpret this program during the transfer.
     ;The program is very simple:
     ;     @1SCNOINC  buffer, #bytes
     ;       SCSTOP
     ;
     LEA     SCSIPseudo(A2),AØ
                   ;point at the pseudo-program block
     MOVE.L  NextAddr(A2),SCParam1(AØ)
                   ;fill in "buffer" with the start address
                   ;of the caller's buffer.
     LSL.L   #1,D6  ;convert #sectors*256 (in D6) to #bytes
     MOVE.L  D6,SCParam2(AØ)  ;fill in "bytes": # of bytes
     LSR.L   #1,D6
     LSR.L   #8,D6     ;restore D6

        ;Do the SCSI call for this iteration.
     MOVEQ   #1,D1     ;assume we're reading (>Ø)
     CMP.B   #aWrCmd,ioTrap+1(A3);are we reading?
     BNE.S   @5     ;yes.
     MOVEQ   #-1,D1              ;no, we're writing. (<Ø).
@5   LEA     SCSIPseudo(A2),AØ   ;point at pseudoprogram
     LEA     SCmd(A2),A1  ;point at SCSI command block
     MOVEQ #6Ø,DØ         ;tick count to wait for completion
     MOVEQ   #1Ø,D2       ;length of command block
     BSR.S   SCSICommon   ;do the xfer
     BNE.S   PrimeErr     ;error in transfer.

     MOVE.L  D6,DØ        ;calculate # sectors  xferred
     ASL.L   #8,DØ        ; * 512, for 512 bytes per sector.
     ASL.L   #1,DØ
     ADD.L   DØ,NextAddr(A2) ;bump buffer pointer.

     ;Let the param block show that we finished.
     MOVE.L  IOByteCount(A3),DØ  ;he asked for this many bytes
     MOVE.L  DØ,IONumDone(A3)    ;we did this many bytes!
     ADD.L   DØ,DCtlPosition(A4) ;bump position pointer, too.
     BRA     DiskOKDone    ;all done!

;This is where we get to if we get an SCSI error during Prime.
;A little intelligence here (retry on certain kinds of errors)
; would be nice.

PrimeErr:
    MOVEQ  #IOErr,DØ ;replace error code with an I/O Error
    BRA    DiskDone  ;and return through our exit routine.
;
; Routine:      SCSICommon
; Arguments:    OurId(A2)       (input)  - our SCSI ID.
;    (AØ)    (input)  - the SCSI pseudoprogram (all set up)
;    (A1)    (input)  - the SCSI command block (6 bytes)
;    (A2)    (input)  - ptr to our locals
;    DØ.L    (input)  - the tick count we're willing to wait
;                       for completion
;    D1.L    (input)  -  sign of byte count we're transferring
;        after the command (Ø if none, + if read, - if write).
;    D2.W    (input)  - Size of the SCSI command block
;        StatWord(A2)     (output) -
;        - returned status from _SCSIComplete, if no err.
;        MsgWord(A2)      (output) -
;        - returned Message from _SCSIComplete, if no err.
;    DØ.W    (output) - result code (error if non-Ø!)
;
; This routine is the common code for an SCSI Operation.
; We set up the command block and the pseudoprogram,
; then call this routine to execute it.
; All the SCSIManager routines we'll be using are stack-based
; functions which return an integer error code. So we "pre-
; push" result word, test it after each command, and pop it
; off after we're done (or, [knock on wood] after an error
; occurs).

SCSICommon
     MOVEM.L D4-D6/A3-A4,-(SP)    ;save regs
     MOVE.L  DØ,D4      ;save tick count in D4
     MOVE.L  D1,D5      ;save byte count in D5
     MOVE.W  D2,D6      ;save command size in D6
     MOVE.L  AØ,A3      ;save the pseudocode ptr
     MOVE.L  A1,A4      ;save the cmd block ptr
     CLR.W   -(SP)      ;reserve space for result code.
                        ;Arbitrate for the SCSI bus
     _SCSIGet           ;"Hello?"
     TST.W   (SP)       ;(did it work?)
     BNE.S   TargetErr  ;(no... give up)

     ;We have the bus... Select our target
     MOVE.W  OurID(A2),-(SP)  ;push our target's SCSI id
     _SCSISelect              ;"Is there anybody out there?"
     TST.W   (SP)             ;(did it work?)
     BNE.S   TargetErr        ;(no... give up)

     ;We've reached our target. Issue the SCSI command
     MOVE.L  A4,-(SP)      ;push address of  SCSI command
     MOVE.W  D6,-(SP)      ; the block is this many bytes long
     _SCSICmd              ;do the command
     TST.W   (SP)          ;(did it work?)
     BNE.S   Cleanup       ;(no... go take care of the error)

; If we have to transfer any bytes, execute the
; pseudo-program to transfer them.
;Since we're going to do an SCSIComplete right away,
;we don't need to check the error
;(because it would only result in a call to SCSIComplete...)
     TST.L   D5            ;any bytes to transfer?
     BEQ.S   Cleanup          ; no, skip this.
     MOVE.L  A3,-(SP)         ;yes, push pseudo-program address
     TST.L   D5 ;Are we reading or writing? (- if writing)
```


© The Definitive MacTutor, Vol. 4

```
    BMI.S   @1 ;br if writing.

     _SCSIRead      ;normal reads only. May be modified
                    ;for other drives
    BRA.S   Cleanup

 @1 _SCSIWrite      ;Do the Write.

;clean up after ourselves, or find out why we got an error.
;(the error so far is in (SP).W - it might be useful to look
; at it).

Cleanup:
    PEA     StatWord(A2)  ;get the status
    PEA     MsgWord(A2)   ; and message words
    MOVE.L  D4,-(SP)      ;tick count we're willing to wait
    _SCSIComplete         ;go get 'em
    TST.W   (SP)          ;(did it work?)
    BNE.S   TargetErr     ;(no. Give up.)

    ;What were the Status bytes?
    MOVE.B  StatWord+1(A2),D0   ;was there an SCSI error?
    BEQ.S   SCSIDone            ;no error, just exit.

  ;At this point, we should look at the "check sense status"
  ;bit in the status byte - if it's set, we could make a
  ;recursive call to SCSICommon to get sense status.
  ;But, since we'd only be returning an I/O error anyway...
  ;
  ;Since this driver always executes fully synchronously,
  ;we'll probably never see the "busy" bit set, but we
  ;could probably retry in that case.

;We got an error in contacting the target, or in SCSIComplete.
TargetErr:
    MOVE.W  #ioErr,(SP) ;if we can't get bus or target.

;Exit, stage left.
SCSIDone:
    MOVE.W  (SP)+,D0                ;pop error code
    MOVEM.L (SP)+,D4-D6/A3-A4       ;restore regs
    RTS       ;and return (D0 and CCR reflect error).
```

Listing 2: SCSI drive formatter/installer

```
\ This program requires the SCSI command definitions
\ printed in V3#3. Only the actual installer code is listed
here.
\ © Nov 1987 J. Langowski / MacTutor

only forth definitions
also mac also assembler

variable numstring 20 vallot
variable mydisk

: input-number numstring 1+ 20 expect
    numstring number? drop ;

: yesno
    BEGIN
    pad 32 expect pad c@
        CASE ascii y OF 1 0 ENDOF
             ascii Y OF 1 0 ENDOF
             ascii n OF 0 0 ENDOF
             ascii N OF 0 0 ENDOF
        1 ENDCASE
    WHILE cr ." Enter y or n - "
    REPEAT
;

: wait ( nticks | #ticks - )
    call tickcount -> #ticks
    BEGIN pause
        call tickcount #ticks -
        nticks >
    UNTIL
;

( SCSI routines follow )
: find.disk ( finds the highest ID SCSI device )
    8 1 do
        SCSIReset drop 60 wait
        SCSIGet drop
        i SCSISelect 0= IF i leave THEN
        loop SCSIReset drop
;

: get.disk find.disk mydisk ! ;

: format 12000 format.blk myDisk @ doscsi 2drop ;

: prep.noattn  ( mode data )
scbuf 20 0 fill
[ hex ]
00000008 scbuf !
00000000 scbuf 4 + !
00000000 scbuf 8 + !
00021000 scbuf C + ! ( page 0; unit attn disable )
[ decimal ]
;

: modenoattn
    prep.noattn
    16 modesel.blk 5 + c!
    120 modesel.blk myDisk @ scbuf 16 doscsi.wb
;

variable ddm 512 vallot
variable dpm 512 vallot
variable driver.block 2048 vallot

: read.cap ( - #blocks bytes.p.blk )
    120 readcap.blk myDisk @ scbuf 8 doscsi.rb
    abort" Can't read capacity" drop
    scbuf @ scbuf 4 + @
;

: show.cap read.cap
    dup . ." byte blocks; total capacity is "
    1024 */ . ." K bytes." cr
;

hex
: create.ddm
    ddm 200 0 fill
    4552 ddm w!
    read.cap ddm 2+ w! ( block size )
             ddm 4 + ! ( # of blocks )
    0 ddm 8 + w! ( device type )
    0 ddm A + w! ( device ID )
    8 ddm C + !  ( first data block )
    1 ddm 10 + w! ( one driver to follow )
    2 ddm 12 + ! ( driver start block )
    4 ddm 16 + w! ( driver is 4 blocks long )
    1 ddm 18 + w! ( and runs on Macintosh =1 )
;

: create.dpm
    dpm 200 0 fill
    5453 dpm w!
    8 dpm 2+ ! ( starting block of partition )
    read.cap drop 8 - dpm 6 + ! ( # of blocks )
    "tfs1 dpm A + !   ( TFS1 signature )
    0 dpm E + !
```

```
;
decimal

: read.ddm
    0 read.blk 2+ w! 0 read.blk 4 + c!
    1 read.blk 5 + c!
    120 read.blk myDisk @ ddm 512 doscsi.r
    2drop
;

: read.dpm
    0 read.blk 2+ w! 1 read.blk 4 + c!
    1 read.blk 5 + c!
    120 read.blk myDisk @ dpm 512 doscsi.r
    2drop
;

: write.ddm
    0 write.blk 2+ w! 0 write.blk 4 + c!
    1 write.blk 5 + c!
    120 write.blk myDisk @ ddm 512 doscsi.w
    2drop
;

: write.dpm
    0 write.blk 2+ w! 1 write.blk 4 + c!
    1 write.blk 5 + c!
    120 write.blk myDisk @ dpm 512 doscsi.w
    2drop
;

: get.sdrv
    "sdrv 128 call getresource
    @ driver.block 2048 cmove
;

: write.sdrv
    0 write.blk 2+ w! 2 write.blk 4 + c!
    4 write.blk 5 + c!
    120 write.blk myDisk @ driver.block 2048 doscsi.w
    2drop
;


.TRAP   _newptr,sys     $A51E
hex 144 CONSTANT SysEvtMask
    308 CONSTANT DQHeader
    6 CONSTANT QTail
    6 CONSTANT DQDrive
decimal

VARIABLE syshp.drvr

: install.driver ( | dstart dlength dbytes pointer - )
    read.ddm
    ddm 18 +  @ -> dstart
    ddm 22 + w@ -> dlength
    dlength 512 * -> dbytes
    dstart 256 /mod read.blk 2+ w! read.blk 4 + c!
    dlength read.blk 5 + c!
    120 read.blk myDisk @ driver.block 512 dlength * doscsi.r
    2drop
    dbytes  MOVE.L (A6)+,D0
    _newptr,sys ( get memory block in system heap )
    MOVE.L A0,-(A6)   -> pointer
    pointer
    IF   driver.block pointer dbytes cmove
         pointer syshp.drvr !
    ELSE ." Not enough system heap for installation." cr
    THEN
;

CODE call.driver
    MOVE.L D5,-(A7)
    MOVE.L (A6)+,D5
    MOVE.L (A6)+,A0
    execute
    MOVE.L (A7)+,D5
    RTS
END-CODE

: mount.scsi
    install.driver
    read.dpm
    SysEvtMask @
    0 SysEvtMask !
    syshp.drvr @ dpm myDisk @ call.driver
    SysEvtMask !
;

: zero.scsi
    DQHeader qTail + @ dQDrive + w@ ( drive # found )
    " JL's Hard Disk" call DIZero
;
```

```
NEW.WINDOW Installer
" SCSI Installer 0.9" Installer TITLE
50 20 300 480 Installer BOUNDS
Document Visible NoCloseBox NoGrowBox Installer ITEMS

1000 6000 TERMINAL Inst

: init.scsi ( | fmt - ) activate
    0 -> fmt
    Installer dup CALL selectwindow
              dup CALL showwindow
              CALL setport
    cr ." Looking for SCSI devices..."
    get.disk
    cr ." SCSI drive found at address " myDisk @ .
    cr show.cap
    cr ." format disk? "
    yesno IF
         cr ." Do you REALLY want to erase this SCSI disk? "
         yesno IF 1 -> fmt
                cr ." Reformatting disk... "
                format
              THEN
         THEN
    modenoattn
    create.ddm create.dpm
    write.ddm  write.dpm
    cr ." Device and partition descriptor maps written. "
    cr ." Reading SDRV resource ... "
    get.sdrv
    cr ." Writing driver ... "
    write.sdrv
    mount.scsi
    fmt if
         zero.scsi
         cr ." Directory zeroed."
         then
    60 wait bye
;

: scsi.go
    Installer ADD
    Installer Inst BUILD
    Inst init.scsi
;

\ use TURNKEY scsi.go filename to create
\ a turnkey application
```

Advanced Mac'ing

Looking At Text From a Different Angle

rot2Edit

John D. Olsen, RPS
Bryan, TX
MacTutor Vol. 4 No. 2

John D. Olsen is a Registered Public Surveyor in the State of Texas with approx 14 years experience in the field of Land Surveying and Civil Engineering. He is currently attending Texas A&M University in pursuit of a B.S. degree in Electrical Engineering. He is currently working at TAMU in the Veterinary Pathology Dept. at the School of Veterinary Medicine. He supports and maintains a network of Macs (approximately 35 machines). He is working on two vertical market applications for the Macintosh. The first is a Scatchard Plot/Ligand Binding program. The second 'on-going' project is a CAD program for Land Surveyors and Civil Engineers involved in land division, development and/or surveying.

"Yes, But Can You Type Upside Down?"

It seems that no matter what we get in the way of library routines, ROM code, standard packages, etc... there is always something that we think 'just should have been included' as a part of the standard Toolbox routines. It seems one of the favorite things to pick on with the Macintosh is the TextEdit routines in ROM.

Although Apple did 'upgrade' the TextEdit routines with the introduction of the new machines most of us feel it is still lacking in several areas. The biggest complaint is usually the 'limit' of 32K per record. Apple did extend TextEdit to handle styles in an integral manner, color text, and with several other goodies that tended to make our lives a little easier.

One of the things I have heard few complaints about and which has admittedly not always been very high up on my wish list is the ability to place rotated text on the screen. It would be especially nice if I were able to type it in the same familiar manner that I use in editors, word processors, etc... I am involved in putting together some vertical market applications and have run up against this problem. For example, I need to be able to put labels on graphs and/or bodies of text in a CAD program. This text necessarily must be able to be read from the right side of the page.

I learned early in my engineering courses that I should always completely define the problem and the givens before even attempting to begin a solution (to be perfectly honest I think we really all begin to try to solve a problem AS we are defining it but it sounds so much more professional to approach it in such an orderly fashion. Back to the problem at hand).

The problem I had required that the text be rotated in even increments of 90 degrees. This is therefore a very specific solution as opposed to general rotation or what I refer to as a 'free rotation' routine; that is, it is not constrained by any particular angular increments as far as the user is concerned.

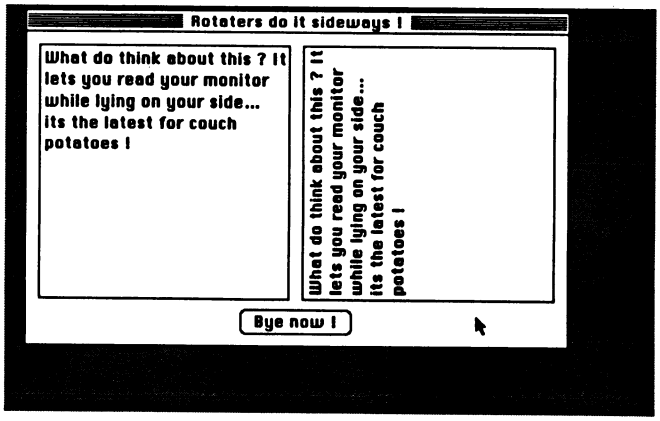


Fig. 1 Our example allows you to edit rotated text!

I normally take the attitude that the general solution is the best solution for the obvious reason of re-usability of the code, adaption to other similar problems and so on. In this case I decided that performance was of the utmost importance and had the feeling that this problems solution would be CPU intensive. I also had a premonition from the start that this program probably would not perform to my satisfaction if written in a high level language.

Pascal is my language of choice when writing sizable programs. I felt rather anxious about the fact that at least a portion of this program might have to be written in Assembly language. Since my only experience to date was in debugging, I had a lot to learn. I was one of those naive souls who only read Assembly code when trying to track down some innocuous piece of code that had the audacity to crash MY PROGRAM and this only after everything else had failed. It seldom helped me solve my problem but as I found the exposure was of great value.

The Plan

The first thing I did was ask around to see if anyone I knew had already solved or attempted to solve this problem. I do not like to re-invent the proverbial wheel unless forced to do so. I talked to Greg and Scott of the MacHax™ Group and they were not aware of any work to date (at least not any 'non-proprietary' stuff) but did express interest and a great deal of encouragement. I checked the bulletin boards and found only one small piece of code that was completely written in Pascal with an unusual approach that used Copybits as the work horse. I tested it and found it took about 7-8 seconds to rotate a 3-4 sq. in. bitMap. This seemed to confirm my feeling that the core routine would have to be in Assembly Language. I also threw away the code I had downloaded.

Having my problem fairly well defined I came up with several possible (?) solutions to the problem. I wanted this routine to behave similarly to DrawString if I could, so that I could simulate standard TextEdit routines by building on this basic module.

The first solution that came to mind was a complete rewrite of the TextEdit routines but done in such a way that the user could select the orientation of the text to be displayed/edited. This had the added advantage of allowing me to add to MY TextEdit the things that I had always wanted. It sounded nice but it was quite a large feat. It also did more (a lot more) than I need it to do. It was like knocking out ant hills with atomic bombs - just a little overkill. I decided to file this idea in the back of my in case the solution I did come up with would be able to be built upon as the 'ultimate TextEdit'.

The next idea was to use offscreen bitMaps and an Assembly Language routine to rotate my text and then CopyBits it back to the desired position. This seemed like the best approach and was chosen as the method I would use. This had the added advantage of being able to rotate graphic objects as well.

Greg Marriott and I discussed pre-rotating fonts as an alternative method to rotating the bitMaps on the fly (FONTs can be thought of as a long narrow bitMap). I could then develop a method for handling the new family of characters. This is an idea well worth exploring but I chose the former solution instead as I felt it was the simpler of the two and had the advantage of being smaller and more general.

What I Did On My Summer Vacation

My programs are written in Pascal with an external procedure written in Assembly Language to do the nitty-gritty of the rotation. I have included several examples of the program to show how it was tested during development along with a couple of ideas for implementation. The programs all function in the same manner although they do different things.

The original program that I wrote for testing the routine simply rotates an ICON. There was actually a predecessor to this program that generated a stripe pattern and rotated it. I quickly found this to be of little use since the pattern was so simple. I was lucky enough to get all but the most trivial problems worked out and got what I expected when I FINALLY got the Assembly code done. (More on that later.)

The second program puts up a modal dialog box that acts as an editor. It has two or four windows depending on the version you choose to run. The two window version works better on the MacPlus as the performance of the four window degrades. The illusion of putting up text in all windows simultaneously works fine in the four window version on the Mac II or an accelerated machine such as my MacPlus with a Novy 020/881 at 16mhz and 4 megs of wide (32 bit) memory.

The third program is an example of using the routine the way I needed it to fill my needs for labels on my graphing and CAD programs. It also has an example of putting in picComments for high resolution output devices. I will discuss the flow and operation of this program. The others are included as examples only.

How It Works

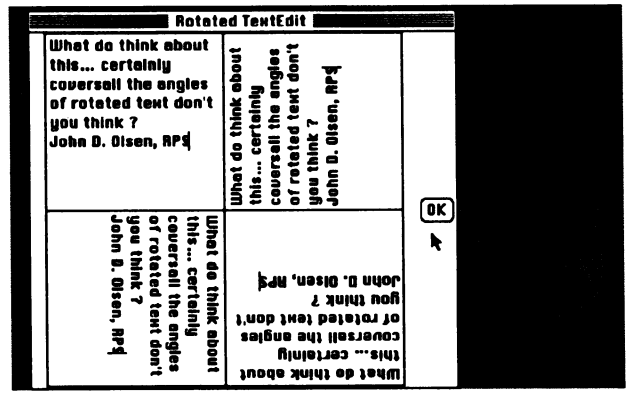


Fig. 2 Expanding text rotation to type upside down!

I prefer the MPW environment for my work although occasionally I do use Lightspeed Pascal for some of my stuff (usually something small/quick). Most of this program is quite straightforward and is all in MPW's Pascal and Assembly Language. I don't use fancy tricks, bend the rules or rewrite anything that I don't have to. See listing 1 to follow along here.

I'll begin at the top of the 'MAIN'. The first thing we do is to Init our environment. This is only done once and some of this is not needed if you use Lightspeed Pascal but is a good thing to keep in practice (it is almost always the same). Then we need to set up our window to draw in, and our grafport. You should notice I'm paranoid about my code getting broken so I check for the screen size and menuBar height (thru screenBits and the low memory global: mBarHeightGlobal = $BAA) then move in from those so that it will work on all systems. I like to try to follow as many of the guidelines as I can, and if someone at Apple (or in this publication) can show me a 'bullet proof' way of doing things you can bet I'll take advantage of it. One of the nice things about MPW is which I keep a bunch of 're-usables' in my Worksheet (like the code to set the window size) that I can copy and paste in every time I start a new application.

The Big Garbonza

At this point I set the font, size and face through toolbox calls. This would more than likely be done either thru a menu, pop-up or dialog box in a real implementation but for now it is hard coded to keep things simple. Now we are ready for our call to then big garbonza, the one we've all been waiting for 'xDrawString'. This is it - this is where it is all really done. (That is except for the Assembly Language stuff and yes I promise I'll get to that if you will just hold your horses a little longer !)

Ok - now back to 'xDrawString'. I decided when I designed the Assembly Language code that I would pass a bitMap to the routine and get one back and that's all. No more parameters. The first stuff is to prepare for this call. After the call we copyBits our new rotated string (bitMap), set up the stuff for the picComments in case we need to copy to MacDraw or print to LaserWriter and cleanup. Thats it.

Counting Hairs

Let's look at it again in detail now.

First we get the FontInfo so we can use some of the code in the record. I need the length and height of the minimum size

bitMap into which the string we want to draw will fit. We call StringWidth() to get the length in pixels. It doesn't get much easier than that.

I then compute the height which will be equal to the ascent height of the font plus twice the descent height. We use twice the descent height so that we get 'white space' all the way around and don't clip any danglies off. I then set my bounding rectangle up with a call to setRect. (These toolbox names are SO handy/ meaningful, don't you think ?) and calculated sizes. Now I have a rect but what I really want is to stuff it into the bitMap record that I am initializing. I can do this with a simple assignment to the 'bounds' field.

Now I call a neat little routine named 'NewBitMapClear' (another one of those re-usables) that will calculate my rowBytes and size for me. It then gets a pointer to the new block of memory (if available) and clears it.

```
newPtrClear FUNC EXPORT
    IMPORT SAVERETA1

    MOVE.L (SP)+,A1
    MOVE.L (SP)+,D0
    _NewPtr   clear
    MOVE.L A0,(SP)
    JMP SAVERETA1

    ENDF
```

The clearing (setting all bits to zero) is quite significant and will be evident why it is important later. Voila our sourceMap is set up.

Next we do some book keeping for later. We get the address of our current grafPort so we can get back to it later. We then go thru the normal gyrations of setting up our offscreen grafPort (with the bitMap we just created) and selecting it. We also make sure the text attributes are the same for this grafPort as our original grafPort since we set our desired text style for it.

Draw That String - Tote That Bale

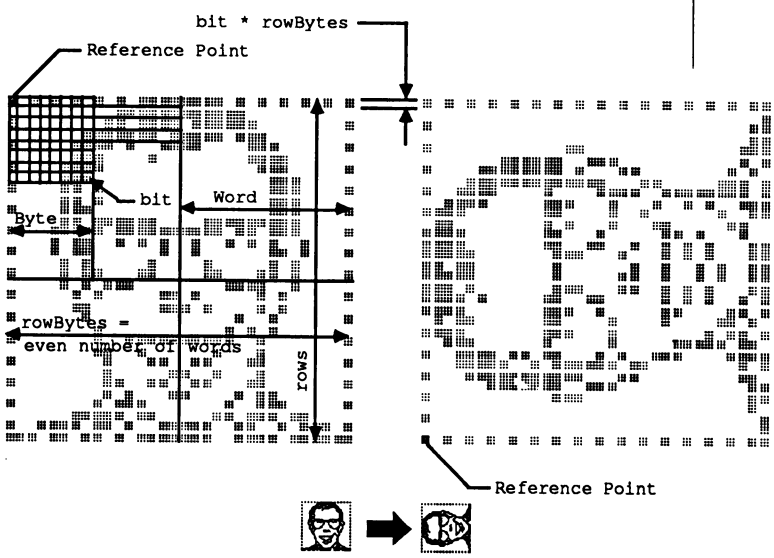


Figure No. 3 - Bitmaps: Before And After

We move from the origin of our new grafPort (0,0) to (0, myFontInfo.ascent) that is we move along he left edge of our bitMap down (in a positive direction) to the ascent height of our font which puts us at the baseline. Then we finally draw our string (keep in mind that this is all 'offscreen')

OK, we've got our offscreen bitMap set up so now I will go ahead and set our port back to where we started:

```
SetPort( origPort );
```

Let's recap to this point:

1. We init all the managers
2. Set up our offscreen bitmap/grafport
3. We copy text attributes from orig->new
4. Move to where we want the string
5. Draw that string offscreen

Ok so we draw our string offscreen, call our assembly language stuff to wave the magic wand. Then we copyBits it to the correct (Original) grafport. The rest of the example code is dedicated to setting up picComments and clean-up. Since these items have been covered in detail before I'll leave them alone for now. Just keep in mind that picComments have become a very important part of all code that has anything to do with images that may be printed. This is so we can take advantage of the high resolution output devices primarily (LaserWriters, plotters, transfer to object drawing programs).

Off To The Land of The Unknown

Now everything was primed and ready to go so we could call our Assembly Language routine. This is the routine that does ALL the real work. It must make a copy of our bitMap ONE BIT AT A TIME (this is not completely true but we'll get to that in a minute). Obviously this is not the kind of thing to do in a high level language because of the tremendous overhead.

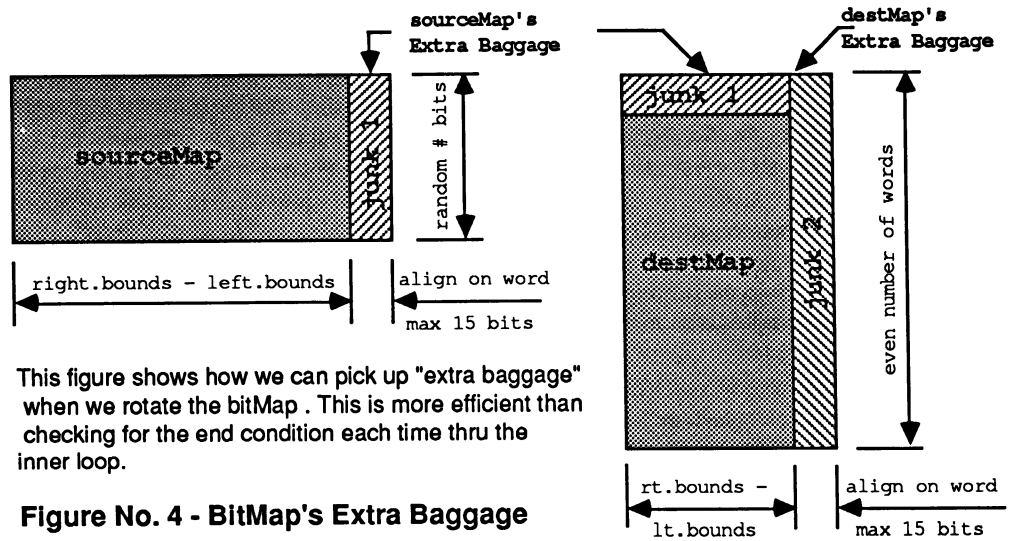
There was only one small complication to doing this procedure. I had NEVER written in assembly language.

With this challenge in front of me and a desire to really understand what I was looking at when I opened up TMON, Nosy or a dumpcode file I was off to my library. I had read all the standards for Macintosh programing and remembered the section in the back Scott Knaster's book "How to Write Macintosh Software" on how to read assembly code. I got it out read it again and realized that I understood a lot of the individual instructions and what they did, the various addressing modes, etc... The thing I didn't understand was HOW/WHERE TO START.

Banging My Head Against The Wall

I decided the only thing to do was to enlist some help.

I went to Greg Marriott of the MacHax™ group and explained my predicament. He agreed to help me. We spent about an hour talking about the same stuff I had just gone over. He also told me that he had a similar problem when he sat down to write his first Assembly



This figure shows how we can pick up "extra baggage" when we rotate the bitMap . This is more efficient than checking for the end condition each time thru the inner loop.

Figure No. 4 - BitMap's Extra Baggage

language routine. We spent the next hour writing a small procedure and talking. That hour was the most enlighting time I had spent in my quest to write this code (or at least it seemed that way).

When I get to a point in my programing where it seems I just can't figure out how to do something it usually helps to either go on to something else or quit for a while. That is good unless you don't really know where to start.

I hope going thru my code will help someone else get past the point where I was hung up. I will discuss the general design and action of this routine and then go thru it step by step. If you have a lot of experience in writing Assembly code the step by step will be quite boring (I suggest you skip on to the next section after *Watching the Grass Grow*).

How & Why of My Asm

The reason we use assembly code is that we want control (in this case at the bit level) without all the overhead. When you get control you are in COMPLETE CONTROL. The Mac is not nearly as forgiving when working at this low level. That means instead of hitting a pot hole in the road you've hit a tree or gone off a cliff.

This routine maps all the bits in the sourceMap to the destMap while rotating the map 90 degrees in a clockwise direction.

I pass the sourceMap and destMap to the asm routine. The routine expects both of these to have been declared as bitMaps but only the sourceMap to have been initialized.

I handle the math involved in computing the new rectangle. The width of the sourceMap is a multiple of words (16 bits or two bytes) wide. That is the definition of rowBytes. The destMaps width, which is not necessarily on an even word, will be set to an even word. Rather than deal with end conditions I leave the height of the DestMap (which is the same as the width of the sourceMap) on an even word boundary.

In the worst case we could end up rotating an extra 15 bit wide strip along the right edge. By doing this extra work we save an extra test for each bit. For small bitMaps the actual 'extra' area is small anyway. For large bitMaps the 'extra' area is a small percentage of the whole.

I setup this bitMap each time thru the routine, therefore the programmer is responsible for disposal of this temporary map. I chose to do this math and memory allocation in the Assembly routine. If the user doesn't like this it would be trivial to remove and handle in high level.

The rotation routine starts at the top left corner of the sourceMap and moves across a word at a time.

I map to the lower left corner of the destMap and move up a word at a time. This is done with 16 words for each word of the destMap. After I reach the top I move over a bit and do it again.

I test each bit for being set and only if true do I get a word from memory and set its corresponding bit. This is possible because I clear memory when I allocate the space for the bitMap. This saves moving memory unnecessarily. This is the most costly portion of the routine because if all bits were set in the sourceMap we would have to get each word from memory and return it 16 times.

Its as simple as that. Nothing real complicated is being done here; it is just time consuming.

The Better Mousetrap

This can all be done in much smaller increments in registers. This will save all the moving in and out and in and out of each word. It should make the performance increase by at least an order of magnitude.

I know of a fellow Aggie here in Bryan/College Station who is working on such a routine and article and look forward to reading about his approach. Until then I will use my stuff.

Watching The Grass Grow

As promised I will go thru the code a chunk at a time for those who wish to avoid the anguish that I went thru.

First we have another little procedure (actually a Function) we should discuss ' newPtrClear'. The only real difference in this and newPtr is that we call this register based routine with the 'clear' parameter. This saves the step of calling EraseRect or something similar to set all bits to zero. This is the code that does that for us:

```
        newPtrClear FUNC EXPORT

        IMPORT SAVERETA1

        MOVE.L (SP)+,A1
        MOVE.L (SP)+,D0
        _NewPtr   clear
        MOVE.L A0,(SP)
        JMP SAVERETA1

        ENDF
```

Lets talk about format since we are getting down to the nitty-gritty. The first thing I should mention about the format of our text file is that it is 'line based'. That is each command/operation is on a single line as opposed to a language like pascal that is procedure based.

The first column is for labels only. Period. Nothing else. If you want to get screwed up put something else there and watch your error list grow out of sight. There is one exception, that being comments, which can go anywhere.

There are several things to remember about comments. The first is that they start with a semi-colon. The next is that they don't have a terminating character so although they can be on the same line they can't be embedded. The only other thing I want to mention is that with Assembly code you should comment everything. Use them excessively. It will save you a lot of time later.

I will get off my soap box now and get on with it. From time to time I will mention some of the things I 'discovered' or found to be pitfalls.

The first part of the code sets up our constants, stackframe and saves our registers so we can get back to the state we were in when we started. This is something you will almost always do.

```
DestMap     EQU 4+4              ; offset to dest bitmap ^
SourceMap   EQU DestMap + 4      ; offset to source bitmap ^
;
; create local stack frame, and save some registers
; make room for dest rect to be passed back
;
    link a6,#0                   ; set up a stack frame
    movem.l d3-d7/a2-a5,-(sp)    ; save reg pascal may need
```

The next couple of lines get us pointing to the bitMaps by getting the address off the stack. This is done by using a displacement off the stack pointer.

```
    move.l  SourceMap(a6),a1     ; point to the source BitMap
    move.l  DestMap(a6),a2       ; point to the dest BitMap
```

Next we will compute the size of our destMap in preparation for filling out its record and getting it some memory. Notice the logical shifts instead of divides and multiplies when working with powers of 2. This save several cycles.

```
    move.w  bounds+bottom(a1),d7    ; get bottom and put in d7
    sub.w   bounds+top(a1),d7       ; bot - top = height, put in
d7
    move.w  d7,d0                   ; save a copy of height
                                ; rnd = (height+15)/16
    addi.w  #15,d7          ; add 15 to the ht, put in d7
    lsr     #4,d7           ; divide by 16
    lsl     #1,d7           ; multiply by 2 = rowBytes of
                            ; dest bitMap
    lsr     #1,d0       ; divide height by 2
    move.w  d0,d2       ; make a copy of height/2 for later
    move.w  bounds+top(a1),d5; get 'top' of source rect
    add.w   d0,d5       ; d5 now contains vert component
                          ; of ctr pt
                        ; compute the width of src rect &
                          ; put in d4
    move.w  bounds+right(a1),d3 ; get right and put in d3
    sub.w   bounds+left(a1),d3  ; right - left = width, put
                              ;   in d3
    lsr     #1,d3               ; divide width by 2
    move.w  d3,d1               ; make a copy of width/2 for
later
    move.w  bounds+right(a1),d4 ; get rt of source rect
    sub.w   d3,d4               ; d4 now contains horiz
                                  ; component of ctr pt
                                ; compute 'top'
    move.w  d5,d3  ; get a copy of ctrPt.v and put
                   ;  in d3
    sub.w   d1,d3  ; ctrPt.v - width/2 = 'top' in d3
    swap d3        ; move 'top' to hiword
                   ; compute 'left'
    move.w  d4,d3  ; get a copy of ctrPt.h and put
                     ; in loword of d3
    sub.w   d2,d3  ; ctrPt.h - height/2 = 'left'
                   ; in loword of d3
;
; we now have 'top' in hiword & 'left' in loword d3
;
                   ; compute 'bottom'
    move.w  d5,d6  ; get a copy of ctrPt.v in of d3
    add.w   d1,d6  ; ctrPt.v + width/2 = 'bottom' in d3
    swap d6        ; move 'bottom' to hiword
                   ; compute 'right'
    move.w  d4,d6  ; get a copy of ctrPt.h and put
                   ; in loword d6
    add.w   d2,d6  ; ctrPt.h + height/2 = 'right'
                   ; in loword d6
```

Note that some of the blocks of comments have been stripped out of the code so we can repeat it here without getting too carried away.

I have already assigned the rect data directly. That is, I put in the structure directly since we can do that here and avoid the overhead of a trap call. This is something that some programmers do not take advantage of. There are two reasons I can think of for this.

The first is that the _SetRect trap is much smaller (fewer lines of code). It also doesn't take much thinking to put it in.

The second and probably more important is that the traps tend to make the code more readable (easier to understand).

The next section of code fills in the rest of the bitMap structure for us.

```
;
; assign data to dest bitMap structure directly
;
    move.w  d7,rowBytes(a2)         ; put in rowBytes direct for
                                      ; dest bitMap
    move.l  d3,bounds+topLeft(a2)   ; put the coord pair in
                                      ; direct
    move.l  d6,bounds+botRight(a2)  ; put the coord pair in
                                      ; direct
    move.w  d7,d0                   ; put rowbytes in d0
    mulu rowBytes(a1),d0        ; mul rowBytes src*dest = d0
    lsl     #3,d0               ; multiply d0 by 8 for size of
                                ; dest bitMap
    _NewPtr clear               ; allocate size of dest bitMap
```

```
                        ; (a2 pts to it)
   move.l  a0,baseaddr(a2)     ; move contents of a0 to
                               ; baseAddr field
                               ; of dest bitMap structure
```

Notice that I repeated the call to _NewPtr with the clear parameter in the code. We could have done this with a subroutine but unrolling the code again saves some time and this is a small piece of code.

The next chunk of code actually gets to the heart of the matter. This is where we start doing all the swapping and testing of bits. We are all set up and ready to roll.

```
   lea  localVars,a0
;
; initialize d0 to contain the pointer to the srcMap
;
   move.w  rowbytes(a1),d0  ; get rowbytes of srcMap
;
; get regs pointing to bits
;
   move.l  (a1),a1       ; a1 pts to BITS of src bitMap
   move.l  (a2),a3       ; a3 pts to BITS of dest bitMap
```

I chose to let the compiler compute the addresses of my local variables for me. This does not cause any additional overhead as it is all done at compile time. I load the address of the beginning of my variables into a0 and access all of them via offsets off this register. I could have used one of the nicer features of MPW that allows the user to build records. The effect is the same. I chose to roll my own because it was something I already understood.

```
   clr.l   d1          ; clear d1
   move.w  d0,d1       ; load rowBytes of srcMap
                       ; (rowBytes of srcMap = ht of
                       ; destMap
   lsl.w   #3,d1       ; multiply by 8 to get ht of
                       ; destMap when rot'd
   mulu.w  d7,d1       ; multiply by rowbytes(destMap) =
                       ; #words in
   sub.w   d7,d1       ; destMap less one row
   add.l   a3,d1       ; add the computed offset to the
                       ; baseAddr of destMap
   move.l  d1,lowerLeft-localVars(a0) ; init lowerLeft
```

The previous section of code computes the lower left corner of the destination bitMap. This is the 'master reference point' for all of the addressing computations for the destMap. All addresses are computed as an offset from this corner. This offset is in the form of a bit and rowbytes times the number of rows. This is not computed each time thru though. It is kept up with counters which are needed anyway.

```
;
; init wordCount to zero
;
   move.w  #0,wordCount-localVars(a0)
                       ; set wordCount to
                       ;zero before we start
;
; init colmLoop counter
;
   move.w  d7,d2       ; get rowBytes(dest)
   lsr     #1,d2       ; divide by 2 - d2 now contains the
                       ; outer counter
   move.w  d2,colmCount-localVars(a0) ; put the max
                       ; value away for reference
   moveq.l #0,d2       ; set d2 to zero since we use it
                       ;  for the counter
colmLoop
;
; this is the outside loop
;
   move.w  #15,d4      ; init colmBitLoop counter to 15
   move.l lowerLeft-localVars(a0),currentWord-localVars(a0)
                       ; init currentWord
colmBitLoop
```

The previous section of code sets up the labels for all the loops and initializes the counters. This is where it got confusing to write. I made a rookie mistake in my first attempts to code this inner section. I tried to code straight thru instead of setting up the loops,then going back and filling them in from the inside out. Once I changed my approach to looking at a couple of lines at a time I had it licked.

We finally get to the point next where we are testing bits and getting a word out of the destMap if true. We then set the appropriate bit and put the word back. Again we only get the word if we need to SET a bit because they are all cleared to start with. This is critical to the performance.

We work completely thru a source word at a time - easy.

```
   move.w  (a1),d3       ; get a word to rotate into d3
   addq.w  #2,wordCount-localVars(a0) ; add 1 for ea word
   moveq.l #0,d6         ; clear d6
   move.w  #15,d6        ; initialize the inside (bit) loop
                         ; counter

innerLoop
;
; we test each bit (15->0 ie. lt to rt) if it is set we worry
; about going out and setting it in destMap otherwise we
; just increment the counters and test the next bit (cause
; we cleared them all at init)
;
   btst d6,d3      ; test bit pointed to in the
                   ; current src word
                   ; by the inner (bit) loop counter
   beq     NoBitToChange
   sub.l   d7,currentWord-localVars(a0)      ; subtract
                                ; rowBytes(destMap) from
                                ; current word
   dbra d6,innerLoop  ; keep looping (thru bits) till d6 = -1
   addq.l  #2,a1      ; point to the next word in srcMap
```

The next piece of code is the bookkeeping section that keeps us moving thru both of the bitMaps in synch. The only thing tricky is the slightly unorthodox use of a dbra which actually is used more as an independent counter than a loop. That is we have a label up top that we get to from two places and the second place gets control only when we exit the dbra 'counter'.

```
   movea.l currentWord-localVars(a0),a2
                         ; get addr of currentWord
   move.w  (a2),d5       ; get currentWord
   bset d4,d5            ; set proper bit in currentWord
   move.w  d5,(a2)
                       ; put the currentWord word back
                       ; into destMap
noBitToChange
doneWithSrcWord
   cmp     wordCount-localVars(a0),d0
       ; cmp wordCount  to rowBytes-are we done with a row ?
```


© The Definitive MacTutor, Vol. 4

```
    bne     rowLoop    ; no - do another word
doneWithRow
    move.w  #0,wordCount-localVars(a0)
                       ; reset wordCount to zero
    move.l  lowerLeft-localVars(a0),currentWord-localVars(a0)
                       ; re-init currentWord
dbra  d4,colmBitLoop   ; go thru the bits (15->0) in the
                       ; destMap - then exit and do it
                       ; again for rowBytes/2 times
doneWithDestWord
    addq.l  #2,lowerLeft-localVars(a0) ; this moves us
                       ; into the next
                       ; 'column' of bits in the destMap
    addq.w  #1,d2      ; add 1 to the colmLoop counter
    cmp     colmCount-localVars(a0),d2 ; are we done yet ?
    bne     colmLoop   ; no - do it again
```

The only thing left to do is our clean-up. Then after our clean-up code is the stuff to set up the local variables (pretty straight forward stuff).

Has The Jury Reached a Verdict?

With the introduction of the new machines life has once again gotten just a little more complicated. We used to be able to think of a one to one relationship for bits and pixels. Life was so easy then (HA!). Anyway, now with the advent of color we have to be careful and make sure we write for the correct machines.

What am I getting to ? This code will work only on the old machines and for the 'old color model'. It will be relatively easy to adapt to the eight bit color mode in the new model but thats another article.

If I were going to give someone some tips before they started to write their first assembly language routine as I just did I would say 'Make it work first'. Now that sounds rather paradoxical I know. What I mean by it is that you should start small and work up. Make you code do the bare minimum first then after it is working enhance it.

I'd Like To Thank The Academy...

If you're real lucky you'll have someone around to ask questions of. I was lucky. I had Greg Marriott. He forced me to learn this stuff - No spoon feeding here. I thank him for that. I know I asked some really dumb questions sometimes. I would also be remiss if I did'nt mention Scott Boyd. He came up with the idea for the quasi-editor rot4Edit in a brainstorming session.

If you don't have anyone around or would like to comment on my code (please don't flame me to bad - this was my FIRST) or if you have a neat idea that I could implement with this code contact me at MCI JOLSEN or AppleLink D0795.

Use this code if you like it and let me know what you do with it.

Listing 1 rotString.p

```
PROGRAM rotString;

{
ooooooooooooooooooooooooooooooooooo
   rotString by John D. Olsen
             for
      MacTutor Magazine
         © Jan 1988
ooooooooooooooooooooooooooooooooooo
}

USES
    {$LOAD MemQukOSToolPack}
    Memtypes, Quickdraw, OSIntf, ToolIntf, PackIntf;
    {$LOAD}

CONST
     picDwgBeg       = 130;
     picDwgEnd       = 131;
     TextBegin       = 150;
     TextEnd     = 151;
     StringBegin     = 152;
     StringEnd       = 153;
     TextCenter      = 154;

TYPE
     TTxtPicRec = packed Record
                  tJus: Byte;
                  tFlip: Byte;
                  tRot: integer;
                  tLine: Byte;
                  tCmnt: Byte;
                  end;
     TTxtPicPtr = ^TTxtPicRec;
     TTxtPicHdl = ^TTxtPicPtr;

     TTxtCenter = Record
                  y, x: Fixed;
                  end;
     TTxtCenPtr = ^TTxtCenter;
     TTxtCenHdl = ^TTxtCenPtr;

VAR
   thehandle : handle;
   windowRect, sourceRect, destRect : rect;
   myWindow : windowPtr;
   sourceMap, destMap, tempMap : BitMap;
   myLabelStr : str255;


FUNCTION NewPtrClear( theSize: size ) : Ptr; EXTERNAL;

PROCEDURE Rotate( srcMap, destMap : BitMap ); EXTERNAL;

PROCEDURE NewBitMapClear( VAR theBitMap : BitMap );
   BEGIN
   WITH theBitMap, bounds DO
      BEGIN
         rowBytes := ((right - left + 15) DIV 16) * 2;
         baseAddr := NewPtrClear(rowBytes * (bottom - top));
         IF MemError <> noErr then baseAddr := NIL;
      END;
   END; (NewBitMapClear)

  PROCEDURE OpenWindow;

    CONST
      mBarHeightGlobal = $BAA;

    VAR
      screen : rect;
      mBarHeight : Integer;
      MemoryPtr : ^Integer;

    BEGIN
      MemoryPtr := Pointer( mBarHeightGlobal );
      mBarHeight := MemoryPtr^;
     screen := screenBits.bounds;

   SetRect( windowRect, screenBits.bounds.left + 5,
screenBits.bounds.top + mBarHeight + 25,
screenBits.bounds.right- 5, screenBits.bounds.bottom - 5 );

   myWindow := NewWindow( NIL, windowRect, 'Text Rotation'
,true , documentProc, windowPtr( -1 ), false, longint( 0 ));
   SetPort( myWindow );
```

```
    END;{OpenWindow}


PROCEDURE xDrawString( myLabelStr: str255 );

   VAR
      myFontInfo: FontInfo;
      strWidth, mapLength :INTEGER;
      mapHeight, xOffset, yOffset : INTEGER;
      sourceMap, destMap : BitMap;
      sourceRect, destRect, myClipRect : Rect;
      MyRgn: RgnHandle;
      origPort, offScrGrafPort : GrafPtr;
      textHandle, centerHandle : Handle;
      Tx, Ty: fixed;

   BEGIN
   GetFontInfo(myFontInfo);
   strWidth := StringWidth(myLabelStr);
   mapLength := ((strWidth - 1) div 16 + 1) * 16;
   mapHeight := myFontInfo.ascent + 2 * (myFontInfo.descent );
   SetRect(sourceRect, 0, 0, mapLength, mapHeight);   { set up
our offscreen bitMap }
   sourceMap.bounds := sourceRect;
   NewBitMapClear( sourceMap );

   GetPort(origPort);
   { create new grafPort to make sure things stay clean }
   offScrGrafPort := GrafPtr(NewPtr(sizeof(GrafPort)));
   OpenPort(offScrGrafPort);
   offScrGrafPort^.portRect := sourceMap.bounds;
   SetPortBits(sourceMap);
   WITH offScrGrafPort^ DO
   BEGIN
      txFont := origPort^.txFont;
      txSize := origPort^.txSize;
      txFace := origPort^.txFace;
   END;

   GetFontInfo( myFontInfo );
   { draw the string we want rotated into our temp bitMap }
   MoveTo( 0, myFontInfo.ascent);
   DrawString( myLabelStr );
   SetPort(origPort);  { set our port back to the original }
   Rotate( sourceMap, destMap );
   {  rotate the offscreen bitmap }

   destRect := destMap.bounds;{ going to draw it ? }
   WITH destRect DO{ offset to 10,10 }
      OffsetRect( destRect, -left, -top );
   OffsetRect( destRect, 10, 10 );
   {set up picComments }
   textHandle := NewHandle(sizeof(TTxtPicRec));
   centerHandle := NewHandle(sizeof(TTxtCenter));
   WITH TTxtPicHdl(textHandle)^^ DO
   BEGIN
      tJus := 2;      {center}
      tFlip := 0;     {no flip}
      tRot := 270;    {rotate 270 degrees}
      tLine := 1;     {single spacing}
      tCmnt := 0;
   END;
   HLock(centerHandle);
   WITH TTxtCenHdl(centerHandle)^^, destRect DO
   BEGIN
      Ty := FixRatio(bottom - top, 2);
      Tx := FixRatio(right - left, 2);
      xOffset := - strWidth div 2;
      yOffset := (myFontInfo.ascent - myFontInfo.descent) div
2;
      x := FixRatio(-xOffset, 1);
      y := FixRatio(-yOffset, 1);
      MoveTo(left + HiWrd(Tx) + xOffset, top + HiWrd(Ty) +
yOffset);
   END;

   PicComment(picDwgBeg, 0, nil);
   PicComment(TextBegin,  sizeof(TTxtPicRec), textHandle);
   PicComment(TextCenter, sizeof(TTxtCenter), centerHandle);
   HUnlock(centerHandle);
   DisposHandle(centerHandle);
   DisposHandle(textHandle);

   SetRect(myClipRect, 0, 0, 0, 0);  { set to 'nada' }
   ClipRect(myClipRect);

   { this just puts the string into the picture }
   DrawString(myLabelStr);

   { now set the clipRect back and draw the rotated bitmap }
   ClipRect(origPort^.portRect);

   CopyBits(destMap, origPort^.portBits, destMap.bounds,
destRect, srcOr, nil);

   PicComment(TextEnd, 0, nil);
   PicComment(picDwgEnd, 0, nil);

   {clean up our mess}
   ClosePort(offScrGrafPort);
   DisposPtr(ptr(offScrGrafPort));
   DisposPtr(sourceMap.baseAddr);
   DisposPtr(destMap.baseAddr);
   SetClip(MyRgn);
   DisposeRgn(MyRgn);
   END;{xDrawString}


{ o o o o o o o o    M A I N    o o o o o o o o o o }
BEGIN
   FlushEvents(everyEvent,0);
   InitGraf(@thePort);
   InitFonts;
   InitWindows;
   InitMenus;
   InitDialogs(NIL);
   InitCursor;
   OpenWindow;
   TextFont(Geneva);
   TextSize(12);
   TextFace([]);
   myLabelStr := 'Looking at text from a different angle';
   xDrawString( myLabelStr );
   REPEAT UNTIL button;
   DisposeWindow( myWindow );
END.
```

Listing 2 Rotate.a Assembly Code

```
ooooooooooooooooooooooooooooooooooo
   Rotate by John D. Olsen
                for
.       MacTutor Magazine
           @ Jan 1988
ooooooooooooooooooooooooooooooooooo
   INCLUDE   'SysEqu.a'
   INCLUDE   'Traps.a'
   INCLUDE   'QuickEqu.a'

newPtrClear FUNC EXPORT

   IMPORT SAVERETA1
   MOVE.L (SP)+,A1
   MOVE.L (SP)+,D0
   _NewPtr   clear
   MOVE.L A0,(SP)
   JMP SAVERETA1
```

```
   ENDF

Rotate PROC EXPORT
;_____________________________
; PROCEDURE  Rotate (sourceMap, destMap : bitMap )
; given a source and destination bitmap rotate it CCW 90°…
; bitmap can be of any size, align on word boundaries (v & h)
;
;_____________________________
;  a0 = result of _newPtr call    |  d0 = size of bitMap
;  a1 = ptr to src bitMap         |  d1 = width/2  (source)
;  a2 = ptr to dest bitMap        |  d2 = height/2 (source)
;  a3 = dest bits                 |  d3 = left | top
;  a4 = copy of dest bits         |  d4 = ctrPt.h
;  a5 = not used                  |  d5 = ctrPt.v
;  a6 = not used                  |  d6 = right | bottom
;  a7 = not used           |  d7 = rowBytes of dest bitMap
;_____________________________
; ***************************************************
; *                                                 *
; *Initialize destination bitMap (destMap), etc…    *
; *                                                 *
; ***************************************************
; parameter offsets
;
DestMap      EQU4+4          ; offset to dest bitmap ^
SourceMap    EQU DestMap + 4 ; offset to source bitmap ^
;
; create local stack frame, and save some registers
; make room for dest rect to be passed back
;
   link      a6,#0           ; set up a stack frame (no bytes)
   movem.l   d3-d7/a2-a5,-(sp)   ; save registers for pascal

   move.l SourceMap(a6),a1 ; point to the source BitMap
   move.l DestMap(a6),a2   ; point to the dest BitMap
;
; compute height of source BitMap rounded up to nearest word
;
   move.w bounds+bottom(a1),d7; get bottom and put in d7
   sub.w  bounds+top(a1),d7; bottom - top = height, in d7
   move.w d7,d0            ; save a copy of height for later
                           ; rnd = (height+15)/16, follows:
   addi.w #15,d7           ; add 15 to the height, put in d7
   lsr    #4,d7            ; divide by 16
   lsl    #1,d7     ; multiply by 2 = rowBytes of dest bitMap
;
; d7 now contains rowBytes of dest bitMap
;
; now compute the destRect for dest bitMap
;
; compute center point of Source rect
; note the center points are not necess. the
; same in the vert dir. relative to srcMap
; because of rounding that follows (i.e. height
; of srcMap is dir. prop. to RowBytes of destMap)

   lsr    #1,d0  ; divide height by 2
   move.w d0,d2  ; make a copy of height/2 for later
   move.w bounds+top(a1),d5; get 'top' of source rect
   add.w  d0,d5  ; d5 now contains vert component of ctr pt
                 ; compute the width of src rect & put in d4
   move.w bounds+right(a1),d3 ; get right and put in d3
   sub.w  bounds+left(a1),d3  ; right - left = width,in d3
   lsr    #1,d3  ; divide width by 2
   move.w d3,d1  ; make a copy of width/2 for later
   move.w bounds+right(a1),d4 ; get right of source rect
   sub.w  d3,d4  ; d4 now contains horiz component of ctr pt
;
; now get center point and compute bounds for dest bitMap
; height and width are reversed now for the 2 bitMaps
; left :=ctrPt.h - height/2
; top  :=ctrPt.v - width/2
; right   :=ctrPt.h + height/2
; bottom:=ctrPt.v + width/2
; SetRect(rect, left, top, right, bottom) put data in direct
to avoid the overhead of a trap call
;
; compute 'top'
   move.w d5,d3  ; get a copy of ctrPt.v and put in d3
   sub.w  d1,d3  ; ctrPt.v - width/2 = 'top' in d3
   swap   d3     ; move 'top' to hiword
                 ; compute 'left'
   move.w d4,d3  ; get a copy of ctrPt.h and put in loword d3
   sub.w  d2,d3  ; ctrPt.h - height/2 = 'left' in loword d3
;
; we now have 'top' in hiword & 'left' in loword d3
; compute 'bottom'
   move.w d5,d6  ; get a copy of ctrPt.v in of d3
   add.w  d1,d6  ; ctrPt.v + width/2 = 'bottom' in d3
   swap   d6     ; move 'bottom' to hiword
                 ; compute 'right'
   move.w d4,d6  ; get a copy of ctrPt.h and put in loword d6
   add.w  d2,d6  ; ctrPt.h + height/2 = 'right' in loword d6
; we now have 'bottom' in hiword & 'right' in loword d6
; assign data to dest bitMap structure directly
   move.w d7,rowBytes(a2)  ; put rowBytes for dest bitMap
   move.l d3,bounds+topLeft(a2)   ; put coord pair in direct
   move.l d6,bounds+botRight(a2)  ; put coord pair in direct
   move.w d7,d0            ; put rowbytes in d0
   mulu   rowBytes(a1),d0  ; multiply rowBytes src*dest = d0
   lsl    #3,d0      ; multiply d0 by 8 for size of dest bitMap
   _NewPtr    clear  ; allocate size of dest bitMap (a2)
   move.l a0,baseaddr(a2)
; move contents of a0 to baseAddr field
; of dest bitMap structure

;_____________________________
;  a0 = baseAddr for our locals
                   |  d0 = rowbytes of srcMap
;  a1 = points to bits of src bitMap
                   |  d1 = width/2  (source)
;  a2 = ptr to dest bitMap |  d2 = height/2 (source)
;  a3 = dest bits     |  d3 = current word srcMap
;  a4 =               |  d4 = copy of width of srcMap
;  a5 = not used      |  d5 = current destMap word
;  a6 = not used      |  d6 = insideLoop count (current bit)
;  a7 = not used      |  d7 = rowBytes of dest bitMap
;_____________________________

; ***********************************************
; *                                             *
; *    Start actual rotation of bitMaps         *
; *                                             *
; ***********************************************
; Assumptions:
; height of destination bitMap = rowBytes of srcMap
; Get the baseAddr for our locals into a0 - this will
; make our program much more readable as we will let
; the assembler do all the addr location calcs for us.
; The addresses will be a fixed number while the
; programs is (locked down) and running, therefore,
; there is no runtime penalty for doing this addr
; calculation
;

   lea localVars,a0
;
; initialize d0 to contain the pointer to the srcMap
;
   move.w rowbytes(a1),d0  ; get rowbytes of srcMap
;
; get regs pointing to bits
;
   move.l (a1),a1       ; a1 pts to BITS of src bitMap
   move.l (a2),a3       ; a3 pts to BITS of dest bitMap

computeInitLowerLeft
```

```
;
; compute the lower left corner of the destMap - this maps to
; the upper left corner of the srcMap and will be used as base
; from which to offset into the dest map when we need to set
; a bit
;
   clr.l  d1         ; clear d1
   move.w d0,d1      ; load rowBytes of srcMap
                     ; (rowBytes of srcMap = ht of destMap)
   lsl.w  #3,d1      ; multiply by 8, get ht of destMap
   mulu.w d7,d1      ; multiply by rowbytes(destMap) = #words
   sub.w  d7,d1      ; in destMap less one row
   add.l  a3,d1  ; add computed offset to baseAddr of destMap
   move.l d1,lowerLeft-localVars(a0) ; init lowerLeft
;
;_____________________________
;  a0 = baseAddr for our locals   |  d0 = rowbytes of srcMap
;  a1 = points to bits of src bitMap |   d1 = SCRATCH
;  a2 = SCRATCH                   |  d2 = colmCounter
;  a3 = dest bits                 |  d3 = current srcMap word
;  a4 = SCRATCH                   |  d4 = colmBitLoop counter
;  a5 = not used                  |  d5 = current word destMap
;  a6 = not used                  |  d6 = insideLoop count
;  a7 = not used              |   d7 = rowBytes of dest bitMap
;_____________________________
;
;  _Debugger;
; init wordCount to zero
;
   move.w #0,wordCount-localVars(a0)
; set wordCount to one before we start
;
; init colmLoop counter
   move.w d7,d2  ; get rowBytes(dest)
   lsr    #1,d2  ; divide by 2 - d2 now contains outer counter
   move.w d2,colmCount-localVars(a0)
                     ; put the max value away for reference
   moveq.l   #0,d2      ; set d2 to zero, use it for counter

colmLoop
; this is the outside loop
;
   move.w #15,d4        ; init colmBitLoop counter to 15
   move.l lowerLeft-localVars(a0),currentWord-localVars(a0)
                  ; init currentWord

colmBitLoop
;
; this is the loop that does the bit counting in the destMap -
; it is really a misuse of the 'dbra' instruction since it is
; not an independent loop but it is a quite cheap method of
; doing our counting
;
rowLoop
;
; loop across each row in the srcMap and keep count each time
; we do a row this 'rowCount' will used to compute the offset
; from 'lowerLeft'
;
   move.w (a1),d3    ; get a word to rotate into d3
   addq.w #2,wordCount-localVars(a0)
                     ; add 1 for each word oooj0ooo
   moveq.l   #0,d6  ; clear d6
   move.w #15,d6     ; initialize inside (bit) loop counter

innerLoop
; we will test each bit (15->0 ie. lt to rt) if it is set then
; we worry about going out and setting it in destMap otherwise
; we just increment the counters and test the next bit
; (cause we ; cleared them all at init)
;
   btst   d6,d3  ; test bit pointed to in the current src word
                 ; by the inner (bit) loop counter
   beq    NoBitToChange
                 ; if bits not set skip to next, we cleared
                 ; them at init of bitMap else set
                 ; the corresponding bit before we bump the
                 ; counter to the next bit
;
; bit needs to be set - get the word from memory, set bit and
; put it back this is very time consuming as each word is
moved
; in/out of memory 16 times and a good place to look at for
; optimization - it probably means a complete
; rethink of the way we are doing this now
;
   nop
   movea.l   currentWord-localVars(a0),a2
                     ; get addr of currentWord
   move.w (a2),d5    ; get currentWord
   bset   d4,d5      ; set proper bit in currentWord
   move.w d5,(a2)    ; put the currentWord word back
                     ; into destMap (memory pt'd to by a3)

   nop

noBitToChange
;
; bit was cleared at init of bitMap so we come here to
; increment the counters - we need to move over 1 bit in the
; srcWord and up 1 row in our destMap (which is always
; subtracting rowBytes
;
   sub.l  d7,currentWord-localVars(a0)
   ; subtract rowBytes(destMap) from current word

   dbra   d6,innerLoop  ; keep looping (thru bits) till d6 = -1
   addq.l #2,a1
   ; point to next word in srcMap - will take
   ; us all the way thru the srcMap as simple as that !
   ; Its the destMap that requires all the dang loops
   ; to get the addr of the corresponding word in the
   ; destMap

doneWithSrcWord
;
; at this point we've just completed a word from the srcMap
;
   cmp    wordCount-localVars(a0),d0
; cmp wordCount to rowBytes-are we done with a row ?
   bne    rowLoop    ; no - do another word

doneWithRow
;
; we just completed a row so add 2 to the count
   move.w #0,wordCount-localVars(a0)
; oooj0ooo reset wordCount to zero

; following use of dbra keeps track of our destMap bit for us
;
   move.l lowerLeft-localVars(a0),currentWord-localVars(a0)
   ; re-init currentWord
   ; set the currentWord to the lowerLeft(current)
   ; lowerLeft is moved over (right) by a word each
   ; time we do a row in the srcMap
   dbra   d4,colmBitLoop   ; go thru the bits (15->0) in the
                           ; destMap - then exit and do it
                           ; again for rowBytes/2 times

doneWithDestWord
;
; were done counting thru the bits of a destMap word -
; at this point we have
; actually finished a column in the destMap. The column
; is a multiple of
; 16 bits wide (and being at least 1 colm) and as tall
; as the destMap is
; (which is also equal to rowBytes of the srcMap)
```


© The Definitive MacTutor, Vol. 4

```
;
   addq.l #2,lowerLeft-localVars(a0)
                        ; this moves us into the next
                        ; 'column' of bits in the destMap
   addq.w #1,d2         ; add 1 to the colmLoop counter
   cmp    colmCount-localVars(a0),d2     ; are we done yet ?
   bne    colmLoop      ; no - do it again
                        ; else we're done
;
; were done so lets do our clean-up/restore
; and go away gracefully
;
cleanUpTime             ; clean-up and get outta dodge
   movem.l   (sp)+,d3-d7/a2-a5    ; restore registers
   unlk   a6            ; clean up stack frame
   move.l (sp)+,a0      ; return address
   add.l  #8,sp         ; pop parameters off stack
   jmp    (a0)          ; bye y'all!

;
; set up our data storage area
;
localVars ; the beginning of our local var storage area
colmCount     ds.w   1
currentWord   ds.l   1
lowerLeft     ds.l   1
wordCount     ds.w   1

   ENDP
   END
```

Listing 3 rot4Edit

```
PROGRAM rot4Edit;
{
oooooooooooooooooooooooooooooooooooo
   rot4Edit by John D. Olsen
idea by Scott Boyd, Greg Marriott
        and John Olsen
             for
      MacTutor Magazine
         • Jan 1988
oooooooooooooooooooooooooooooooooooo
}
uses
   {$LOAD pinterfaces.dump}
   MemTypes,QuickDraw,OsIntf,PasLibIntf,
   ToolIntf,PackIntf,IntEnv,CursorCtl;
type
   integerPtr = ^INTEGER;
var
   item: Handle;
   itemHit, invertItem : INTEGER;
   map1, map2, map3, map4 : bitMap;
   rect1, rect2, rect3, rect4 : rect;


FUNCTION NewPtrClear( theSize: size ) : Ptr; EXTERNAL;

PROCEDURE Rotate( srcMap, destMap : BitMap ); EXTERNAL;

PROCEDURE NewBitMapClear( VAR theBitMap : BitMap );
   BEGIN
   WITH theBitMap, bounds DO
      BEGIN
         rowBytes := ((right - left + 15) DIV 16) * 2;
         baseAddr := NewPtrClear(rowBytes * (bottom - top));
         IF MemError <> noErr then baseAddr := NIL;
      END;
   END;

FUNCTION filterProc(theDialog: DialogPtr; VAR theEvent:
EventRecord; VAR itemHit: INTEGER): BOOLEAN;
      CONST
         EnterKey = $03;
         BackspaceKey = $08;
         TabKey = $09;
         ReturnKey = $0D;
         ClearKey = $1B;
         DeleteKey = $7F;

         {bits of the event modifiers long word}
         CommandBit = 8;
   VAR
      itemNum: INTEGER;
      kind: INTEGER;
      box: Rect;
      charCode: INTEGER;
      theChar: Char;
      dialog: DialogPtr;
   noteDialog : DialogPtr;
   itemType : integer;

   BEGIN
   filterProc := FALSE;

   CASE theEvent.what OF
      nullEvent:
         BEGIN ( 2, 3, 4, 5 )
            GetDItem(theDialog, 2, kind, item, box);
            GetDItem(theDialog, 3, kind, item, rect2);
            GetDItem(theDialog, 4, kind, item, rect3);
            GetDItem(theDialog, 5, kind, item, rect4);
            CopyBits( thePort^.portBits, map1, box,
map1.bounds, srcCopy, nil);
            Rotate( map1, map2 );
            Rotate( map2, map3 );
            Rotate( map3, map4 );
            CopyBits( map2, thePort^.portBits, map2.bounds,
rect2, srcCopy, nil);
            CopyBits( map3, thePort^.portBits, map3.bounds,
rect3, srcCopy, nil);
            CopyBits( map4, thePort^.portBits, map4.bounds,
rect4, srcCopy, nil);
            DisposPtr( map2.baseAddr );
            DisposPtr( map3.baseAddr );
            DisposPtr( map4.baseAddr );
         END;
      keyDown, autoKey:
         BEGIN
         charCode := BAND(theEvent.message, charCodeMask);
         IF charCode IN [EnterKey] THEN
            BEGIN
            itemHit := 1;
            filterProc := true;
            EXIT(filterProc);
            END;
         IF charCode IN [ClearKey] THEN
            BEGIN
            IF theEvent.what = keyDown THEN
DlgDelete(theDialog);
            theEvent.what := nullEvent;
            EXIT(filterProc)
            END;
         IF BTST(theEvent.modifiers, CommandBit) THEN
            BEGIN
            theChar := CHR(charCode);
            CASE theChar OF
               'X','x': IF theEvent.what = keyDown THEN
DlgCut(theDialog);
               'C','c': IF theEvent.what = keyDown THEN
DlgCopy(theDialog);
               'V','v': IF theEvent.what = keyDown THEN
DlgPaste(theDialog);
               OTHERWISE ;
            END;
            theEvent.what := nullEvent;
            EXIT(filterProc);
```

```
            END;
         END;
   END;
   END;

PROCEDURE EditThatSucker;
type
   strHandle = ^strPtr;
   strPtr = ^Str255;
var
   noteDialog : DialogPtr;
   itemType : integer;
   item : handle;
   box : rect;
BEGIN
   {Bring up the window}
   noteDialog := GetNewDialog( 1000, NIL, windowPtr(-1) );
   showWindow(noteDialog);
   SetPort(noteDialog);
   getDItem(noteDialog, 2, itemType, item, box);
   map1.bounds := box;
   NewBitMapClear( map1 );
   InitCursor;
   repeat
      ModalDialog(@filterProc, itemHit);
   until itemHit = 1;
   DisposDialog(noteDialog);
END;{EditThatSucker}

BEGIN {main}
   InitGraf(@thePort);
   BEGIN
      InitFonts;
      InitWindows;
      InitMenus;
      TEInit;
      InitDialogs(nil);
   END;
   invertItem := 1;
   EditThatSucker;
END.
```

Listing 4 Resource File for rot4Edit

```
#include "Types.r"

type 'xRot' as 'STR ';

resource 'xRot' (0) {
   "Sample Rotation Application - Version 1.0 by John D.
Olsen,RPS of Double Centre Surveying"
};


resource 'DITL' (32723) {
   {  /* array DITLarray: 5 elements */
      /* [1] */
      {140, 326, 160, 354},
      Button {
         enabled,
         "OK"
      },
      /* [2] */
      {3, 16, 147, 160},
      EditText {
         enabled,
         "What do you think about this… certainly covers"
         "all the angles of rotated text don't you th"
         "ink ?"
      },
      /* [3] */
      {3, 165, 147, 309},
      EditText {
         enabled,
         ""
      },
      /* [4] */
      {152, 165, 296, 309},
      EditText {
         enabled,
         ""
      },
      /* [5] */
      {152, 16, 296, 160},
      EditText {
         enabled,
         ""
      }
   }
};

resource 'DLOG' (1000, "Rotated Text Dialog") {
   {41, 16, 341, 372},
   documentProc,
   visible,
   noGoAway,
   0x0,
   32723,
   "Rotated TextEdit"
};
resource 'BNDL' (128) {
   'xRot',
   0,
   {  /* array TypeArray: 2 elements */
      /* [1] */
      'FREF',
      {  /* array IDArray: 1 elements */
         /* [1] */
         0, 128
      },
      /* [2] */
      'ICN#',
      {  /* array IDArray: 1 elements */
         /* [1] */
         0, 128
      }
   }
};

resource 'FREF' (128) {
   'APPL',
   0,
   ""
};

resource 'ICN#' (128) {
   {  /* array: 2 elements */
      /* [1] */
      $"FFFF FFFF 8000 8001 A400 807D AA00 8001"
      $"AE00 8039 AA00 8015 A000 8039 8000 8001"
      $"8000 8001 8000 8001 8000 8001 8000 8001"
      $"8000 8001 8000 8001 8000 8001 8000 8001"
      $"FFFF FFFF 8000 8001 8000 8001 8000 8001"
      $"8000 8001 8000 8001 8000 8001 8000 8001"
      $"8000 8001 9C00 8005 A800 8055 9C00 8075"
      $"8000 8055 BE00 8025 8000 8001 FFFF FFFF",
      /* [2] */
      $"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
      $"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
      $"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
      $"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
      $"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
      $"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
      $"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
      $"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
   }
};
```

The Mac Hacker

QuickTrap Routines Bypass Trap Dispatcher

Mike Morton
University of Hawaii
MacTutor Vol. 4 No. 3

Bypassing the ROM trap dispatcher

In an article a while back, I covered the basics of bypassing the Macintosh trap dispatcher to call ROM routines directly, to speed up calls to the Toolbox and OS. In this article, I'll present a set of subroutines which implement this technique in a practical way.

The package is written in MPW assembler, and should be easily callable from any of the MPW languages. It's short and should be portable to other development systems. It also includes a "fail-soft" feature, in case it turns out not to work on some future Macintosh.

A quick review

Programs call the Macintosh Toolbox and Operating System routines by executing "illegal" instructions, which are handed to the trap dispatching code in the ROM. In addition to the time it takes for the 680x0 processor to recover from the emotional trauma of this illegal instruction, the dispatcher must fetch the offending instruction, decode it, and call the routine it specifies. This is very general, since it "hides" ROM locations from the application, but it's also slow.

With the GetTrapAddress routine, you can calculate the address of a ROM routine just once each time your application runs. Calling that address directly can save you a lot of time, with very little cost in generality.

What does the dispatcher do?

Here's the code for the dispatcher in my MacPlus ROM. Your Mac may have something a little different, but all existing Macs seem to be similar in principle. The dispatcher, at address $401F52 in my ROM, disassembles to:

```
disp:
    SUBQ.L    #2, SP              ; add 2 bytes above CCR
    MOVEM.L     D1-D2/A2, -(SP)   ; save 12 bytes of regs
    MOVE.L    12+4(SP), A2        ; get PC of trap word
    MOVE.W    (A2)+, D2           ; get A-trap word
    MOVE.L    A2, 12+4(SP)        ; restore updated PC
    MOVE.W    D2, D1              ; copy trap word to D1
    ANDI.W    #$01FF, D2          ; get just trap number
    CMPI.W    #$A800, D1          ; trap or OS?
    BLO.S     doOS                ; jump if OS
    LEA       $0C00, A2           ; point->Toolbox dispatch
    LSL.W     #2, D2              ; scale number->longwords
    MOVE.L    (A2,D2.W), 12(SP)    ; copy address to stack
    CMPI.W    #$AC00, D1          ; "auto-pop" bit set?
    MOVEM.L   (SP)+, D1-D2/A2     ; restore regs; leave CCR
    BLO.S     callTB              ; skip if "auto-pop" off
    MOVE.L    (SP)+, (SP)         ; RTS to caller, not glue
tBox:       RTS                   ; "call" Toolbox routine

doOS:
    LEA       $0400, A2       ; point to OS dispatch
    BCLR      #8, D2          ; clear&test "keep A0" bit
    BNE.S     OSa0            ; skip to allow A0 returned
    LSL.W     #2, D2          ; scale number->longwords
    MOVE.L    (A2,D2.W), A2   ; fetch OS routine address
    MOVEM.L     A0-A1, -(SP)  ; save regs (incl A0)…
    JSR (A2)                  ;  …call OS routine…
    MOVEM.L     (SP)+, A0-A1  ;   …and restore OS regs
OSrt:
    MOVEM.L     (SP)+, D1-D2/A2; restore OUR regs
    ADDQ.W    #4, SP          ; ignore stacked CCR
    TST.W     D0              ; preset CCR on result
    RTS                       ; and return

OSa0:
    LSL.W     #2, D2          ; scale number->longwords
    MOVE.L    (A2,D2.W), A2   ; fetch OS routine address
    MOVE.L    A1, -(SP)       ; preserve A1, *not* A0…
    JSR       (A2)            ;  …call OS routine…
    MOVE.L    (SP)+, A1       ;   …and restore A1
    BRA.S     OSrt            ; clean up with common code
```

[An aside: This is the first piece of ROM code I ever read, and I still think it's a great example of tight 68000 coding. It's tighter on the Mac II, with indirect addressing available. I can't see any way to make it faster; can anyone spot a way to save a few bytes, though?]

Besides figuring out which routine to call (using the Toolbox dispatch table at $0C00 or OS table at $0400), the dispatcher also does some other important things. For Toolbox traps, it discards the return address if the "auto-pop" bit is set — this is useful for "glue". And for OS traps, it preserves D1, D2, A1 and A2, and sometimes A0. For OS traps, it also passes the low nine bits of the trap number to the routine, in D1,

Our task is to make a trap "dispatcher" which does all this, but is much faster. Note, for instance, that the new code must still pass the trap number in D1.w — I believe this is how some routines test for flag bits set in the word. (For instance, CmpString has a bit to specify if the comparison is case-sensitive.)

Hey, wait a minute! Isn't it a bad idea to know how one ROM routine (the dispatcher) communicates with all the others? Isn't code which depends on this interface likely to fall apart when the Mac III hits the streets? Well, first of all, it'd be awfully hard for Apple to change hundreds of routines. But more importantly, there's a way to back out gracefully. Trust me; we'll get to it…

An application's view of the QuickTrap routines

The fundamental speedup is to get rid of the dispatcher, and have one "quick trap" routine for every real routine you'd like fast access to. For instance, if your program does a lot of SetPort calls, you can easily create "qtSetPort", which has exactly the same interface and does the same thing, only faster. As you might guess, each qt*xxx* routine caches the address for its routine.

Once, at the beginning of your application, you must call qtEval, which "evaluates" each address and stores it. If you don't

call it, everything will still work — this is related to the fail-soft scheme.

Other than this, everything works the same as old-style trap routines.

Caching problems

Imagine that you spend a lot of time doing FrameOval calls to draw circles on the screen, and would like to speed this up. (Actually, I'm sure the trap time is insignificant compared to the drawing time; this is just an example.) You install "qtFrameOval" and call it instead everything works great.

Now your friend gives you this neat, public-domain desk accessory which causes all ovals to be drawn on your screen with smile-faces in them. [Any takers to write this, by the way? You could call it The Smiling Moose...] It does this by altering the FrameOval trap to call it. But since your application never executes that trap, its ovals are drawn unmolested. How can you make sure your ovals are happy?

The answer is to call qtEval at the right times — not just at initialization but whenever you suspect someone has installed a replacement trap routine. Since the qtxxx routines are supposed to "cache" the real addresses, they must track new address when they're installed, or the cache becomes "stale".

One way to do this is to call qtEval every time you regain control from a desk accessory, each time you regain control from Switcher or Multifinder, and each time you invoke an FKEY. Perhaps you'd also have to call it for every SystemTask call. And of course you must call it if your application does any SetTrapAddress calls for the relevant traps. In short, whenever anyone could have changed trap addresses, refresh the cache.

A simpler approach is to change the SetTrapAddress trap by installing a prefix routine which sets a flag in your globals that re-evaluation is needed. If DAs, FKEYs, etc., play by the rules and use SetTrapAddress calls, nobody can make the trap tables get out of sync with your cached addresses.

It's tempting to call qtEval in your idle-loop as a heavy-handed way to make sure it's done often enough. I suspect this is a bad idea — it can cause seemingly random bugs.

One other way: if you use, for instance, qtFrameOval only in some code which doesn't relinquish control, call qtEval once before each time you enter that code. Remember that qtEval isn't all that speedy — it must call GetTrapAddress for every qtxxx routine.

Reasons not to use these routines

Because the routines are JSR'd to, they take up four bytes instead of two. This is no big problem for most applications, but don't change all your calls.

When you're debugging, commands to break on traps don't work, since your application is not executing trap instructions. You can force these traps to occur by disabling the caching; see below for details.

The routines use impure code. You must make sure you put them in a segment which is locked in memory.

Which traps should you replace?

Remember that many traps take so much time that the dispatch isn't worth improving. Others do next to nothing, and speed up a lot. In early use of these routines at Lotus, we estimated about thirty routines were worth replacing. In the OS world, things like BlockMove and UprString were included. Routines which just twiddled handles are also important, like HLock, HUnlock, HPurge, HNoPurge, and GetHandleSize. Among the Toolbox routines, things like MoveTo and SetPort seemed to help.

Even if a routine is slow, it may be worth tweaking if it's called a lot. We got measureable improvements substituting for CharWidth, DrawString, StringWidth, and SystemTask.

You can also replace package calls, which is kind of a pain. If you want to change all the FP68K traps to qtFP68K, you have to change Apple's include files, since each of the SANE macros invokes the trap. Another solution is to just redefine FP68K to be a macro to JSR to the qtxxx routine. But then you have to define a trap like "myFP68K" which still expands to the A-line trap — this is because the qtxxx routine must have a copy of the trap word.

How much does it help?

As the TV diet ads say, results vary directly with how closely you stick to the plan. Average performance in a large Macintosh product at Lotus was improved by about 5%. A couple of heavily CPU-bound loops were improved by 15%. These aren't huge gains, but considering that they took only a day or so of work to install in a very large program, they're pretty good.

When does the warranty run out?

OK, it's time to face the music. If these routines dive directly into the ROM, they may someday dive into ROM routines in a new machine which expect different parameters. (For more on this topic, see Macintosh Technical Note #110.) Or even if the ROM doesn't change, some caching problem may come up if your application's users use some odd way of altering trap addresses and making your cache stale.

The initialization routine qtEval can be easily disabled by modifying resources. For instance, when a user calls to complain that some FKEY or DA doesn't work with your application, you can quickly change a copy of the application to disable address caching and test if that's the problem. If it is the problem, you can either distribute the altered application or tell power users how to edit the resources to alter the copy they already have.

The resource used to control caching is QTRP 257. The format is simple: if the resource is present and the first word is zero, caching is enabled. To turn off caching, just remove this resource under Resedit (or renumber it, to easily restore caching). Remember that programmers may want to disable caching for certain types of debugging when they want to "see" traps under a debugger.

In a future format, a non-zero first word could signal that the resource contains a list of specific traps to be enabled/disabled.

In short, it's easy to experimentally turn off this hackery to check if it's causing problems, and easy to turn it off permanently if it is. In tests at Lotus, an application with caching disabled ran less than 1% slower than one which executed traps in the first place. This is the cost of calling a qtxxx routine, which in turn must do the xxx trap anyway because caching is off.

Notes on the code

The routines are intended to be pretty simple; I'll walk

through them and point out a few things.

Code caching: While this stuff works fine on a Mac II, I believe it ought to flush the 68020's code cache after patching itself. Any recommendations from 68020 gurus out there?

Layout: The Toolbox and OS routines are laid out with symbols defining offsets in them. This is so they can be patched; the symbols must stay in sync with the layout.

Toolbox routines: These start out looking a lot like "glue" routines for a higher-level language — they use the "auto-pop" bit, so their return is ignored and the trap returns straight to the application routine which called qt*xxx*. This trap word, plus four bytes of slop, is patched to be JMP <trap address>. Also, before the entry point there's another copy of the trap word, in case qtEval is called more than once.

OS routines: These are more complicated. In their simple form, they execute the trap and return, because there is no "auto-pop" bit for OS calls. After being patched, they do just what the dispatcher does: save registers, set up D1 and D2 with the trap number, JSR to the routine, restore registers, test D0.w, and return. Note that the registers saved and restored depend on the trap word — if bit 8 is set, then A0 is included in the registers saved.

Bit-coding: The OS routines hard-wire the trap number passed in D1 and D2. If you want to call, for example, NewHandle with the "clear" bit set, you must define two routines: qtNewHandle and qtNewHandleClear (or whatever you want to call them). This is necessary because your JSR qtNewHandle can't communicate whether it wants the "clear" bit set — that's something normally encoded in the trap word.

Adding routines: The qtTool and qtOS macros do all the work for you. For each one, supply the name of the qt*xxx* routine you want to define and the _*xxx* name of the trap it's going to handle.

Using the routines

Pick some segment which won't leave memory and change the "SEG" directive to specify it.

Assemble the routines and link them into your program as normal. If you're using a higher-level language, declare the qt*xxx* routines to have exactly the same calling interface as the _*xxx* trap, except that they're defined externally instead of invoking in-line trap words.

Remember to call qtEval once at startup. And if you want to avoid the cache getting stale, use one of the strategies described above to decide when to call qtEval again.

If you're using a language which uses glue, you may not be able to easily do this. Write to your language developer and pester them to do it for you...

Comments? Improvements? Letter bombs?

I'd be interested to know how these routines work, how easy they are to install in various development environments, and what kind of performance improvements you see from using them. Drop me a line at P.O. Box 11378, Honolulu, HI 96828.

Since this stuff is stretching the ROM in ways it wasn't meant to be stretched, I'd also appreciate hearing about the technique in general. Do you think it's safe? Can you suggest a better way? And if you have improvements, send them in to

MacTutor...

```
; Macintosh Toolbox and OS-trap bypass routines.
; Copyright © 1987 Michael S. Morton
;
; History: 10-May-87 - MM - Initial version.
;      22-Oct-87 - MM - Neatened for publication.

   BLANKS ON
   STRING ASIS
   PRINT  OFF
   LOAD   'tlAsmSyms.sym'
   PRINT  ON

   ; Impure code! Should be in a segment which is locked in memory.
   SEG 'LOCKDSEG'    ; *** change to a locked segment ***
;-------------------------------
; Each Toolbox routine starts out life as:
;      2 <pure copy of trap word>
;  entry: 2 <trap word with auto-pop bit>
;      4 <four bytes unused>
; The evaluator changes this to:
;      2 <pure copy of trap word>
;  entry: 6 JMP    <actual address>
;
; Either form is callable with a JSR because the former
; includes the "auto-pop" bit, so the Toolbox routine returns
; to its caller's caller.  Offsets are:
tTrap:    EQU 0      ; pure copy of trap
tJump:    EQU 2      ; JMP xxx.L instruction
tAddress: EQU 4      ; address to jump to
tLength:  EQU 8      ; length of one block

;  The "qtTool" macro generates code for one "qt" Toolbox routine.
; args:     routine -   Name for routine.
;                       Typically "qt" plus trap name.
;             trap    - Name of the trap, eg, "_MoveTo".

   MACRO
   qtTool
      EXPORT &Syslst[1]; define the "qtXXX" routine globally
      &Syslst[2]       ; first, a pure copy of the trap word
      &Syslst[1]&Syslst[2]  autoPop
                       ; entry: if not overwritten, just trap
      ds.b   4         ; reserve room for overwriting
   ENDM
;-------------------------------
; Each OS trap bypass routine is 28 bytes long.
; The unevaluated routine is:
;      2 <pure copy of trap word>
;  entry: 2 <trap word>        ; only this part…
;      2 RTS                   ; …gets executed before eval
;      4 MOVE.W  #<xxx>, D1    ; trap word patched here
;      4 MOVE.W  #<xxx>, D2    ; trap number patched here
;      6 JSR     xxx.L         ; routine address patched here
;      4 MOVEM.L (SP)+, D1-D2/A1-A2; register list patched here
;      2 TST.W   D0            ; set condition codes
;      2 RTS                   ; and return
; After the evaluator is done, the routine becomes:
;      2 <pure copy of trap word>
;  entry: 4 MOVEM.L D1-D2/A1-A2, -(SP)  ; (saves A0 too,
;                                       ;  if bit 8 set)
;      4 MOVE.W  #<trapword>, D1        ; get trap word in D1
;      4 MOVE.W  #<trapword & $01FF>, D2 ; …and trap number
;      6 JSR     xxx.L                  ; call the routine
;      4 MOVEM.L (SP)+, D1-D2/A1-A2  ; (gets A0 too, if bit 8 set)
;      2 TST.W   D0                   ; set condition codes
;      2 RTS                          ; and return

oTrap:    EQU 0      ; original trap word
oSave:    EQU 2      ; for MOVEM.L xxx, -(SP) to go
oTrapWord:EQU 8      ; for trap word in MOVE.W #xxx, D1
oTrapNum: EQU 12     ; for trap number in MOVE.W #xxx, D2
oAddress: EQU 16     ; address to jump to
```

```
oRestore: EQU 22      ; for second copy of MOVEM regs list
oLength:  EQU 28      ; length of one block

;   The "qtOS" macro generates code for one "qt" OS routine.
;
; args:     routine -   Name for trap routine.
;                       Typically "qt" plus trap name.
;           trap    -   Name of the trap, eg, "_GetHandleSize".

   MACRO
   qtOS
      EXPORT &Syslst[1]; define the "qtXXX" routine globally
      &Syslst[2]      ; pure copy of trap word
      &Syslst[1]&Syslst[2]
                      ; entry: if not overwritten, just trap…
      RTS             ; …and return
      MOVE.W #$5555, D1; get trap word in D1, for OS routine
      MOVE.W #$5555, D2; …and trap number
      JSR $55555555   ; leave space for a longword address
      MOVEM.L (SP)+, D1-D2/A1-A2
                      ; assume A0 not in the register list
      TST.W  D0       ; set condition codes
      RTS             ; and return
   ENDM

;_____________________________
; Resource type and ID for the flag used to disable the trickery.

qtrpType: EQU 'QTRP'        ; resource type for flag
qtrpId:   EQU 257           ; resource ID for flag


;   qtEval
;
; description:
;  Update the routines so they jump directly into
;  the ROM, or wherever. This routine should be called
;  at startup, and each time the application
;  thinks anyone has (or might have) called SetTrapAddress.
;
; uses:   (no registers)
qtEval:   PROC   EXPORT
; Stuff used in patching together routines:
jmpInst:  EQU $4EF9           ; opcode word of JMP xxx.L
OSregs: REG   D1-D2/A1-A2     ; registers saved by OS dispatcher
OSregs2:REG   D1-D2/A0-A2     ; registers saved when bit 8 is zero

   MOVEM.L D0-D2/A0-A2, -(SP)  ; save caller's registers
   ; First, see if we've already been told not to do our thing:
   LEA qtEnabled, A2       ; point to the flag
   TST.B  (A2)             ; have we snuffed it already?
   BEQ evalEnd             ; yes: nothing to do
   ; Second, decide if the resource flag allows us to map/cache.
   SUBQ   #4, SP           ; make room for function result
   MOVE.L #qtrpType, -(SP) ; pass the type…
   MOVE.W #qtrpId, -(SP)   ; …and ID
   _GetResource            ; try to find our flag
   MOVE.L (SP)+, A0        ; pop result…
   MOVE.L A0, D0           ; …and test for NIL
   BEQ.S  evalTurnOff      ; no such thing? go flag this and exit
   MOVE.L (A0), A1         ; deref. handle; point to rsrc with A0
   MOVE.W (A1), D2         ; pick up first word, to check later
   MOVE.L A0, -(SP)        ; pass handle…
   _ReleaseResource        ; …and get rid of it
   TST.W  D2               ; now check - did rsrc start with zero?
   BNE.S  evalTurnOff   ; …no: we don't yet do selective disable

   ;  Nothing forbids hackery.  Evaluate all the
   ;  toolbox bypass routines.

   LEA qtToolStart, A1     ; point to first routine
   LEA qtToolEnd, A2       ; point to just after last routine
   MOVE.W #jmpInst, D1     ; get a JMP xxx.L instruction
   BRA.S  toolEnd          ; check for no routines

toolLp:   MOVE.W tTrap(A1), D0 ; pick up the trap number in D0.w
   _GetTrapAddress newTool ; ask where this routine lives
   MOVE.L A0, tAddress(A1) ; store address first, THEN…
   MOVE.W D1, tJump(A1)    ; …the JMP, so routine's always OK
   ADDQ   #tLength, A1     ; advance to the next routine

toolEnd:CMP.L A2, A1       ; at (or past) end of toolbox routines?
   BLO.S  toolLp           ; nope: go evaluate another one

   ; Evaluate all the OS bypass routines.
   LEA qtOSStart, A1       ; point to first…
   LEA qtOSEnd, A2         ; …and to just after last
   BRA.S  osEnd            ; handle degenerate case

osLoop: MOVE.W   oTrap(A1), D0 ; pick up trap number
   MOVE.W D0, D2           ; copy it for later use (BTST, etc.)
   _GetTrapAddress newOS   ; find where the routine lives
   MOVE.L A0, oAddress(A1) ; save routine address in JMP xxx.L

   MOVE.W D2, oTrapWord(A1); fill in MOVE.W #trapword, D1
   AND.W  #$01FF, D2       ; get just the trap number
   MOVE.W D2, oTrapNum(A1) ; and store in MOVE.W #trapnum, D2

   ; Decide whether the saved registers include A0.
   MOVE.L OSent, D0     ; assume we want usual registers saved…
   MOVE.L OSexit, D1    ; …and restored
   BTST   #8, D2        ; but should we save A0, too?
   BNE.S  osLp1         ; nope: OSent's registers are fine
   MOVE.L OSent2, D0    ; yep: use reg list which includes A0
   MOVE.L OSexit2, D1   ; …and ditto for one which saves A0
osLp1: MOVE.W D1, oRestore(A1) ; store register list for restore
   MOVE.L D0, oSave(A1)    ; lastly, get rid of 1st trap
   ADD #oLength, A1        ; advance to the next routine

osEnd: CMP.L  A2, A1       ; at the end?
   BLO.S  osLoop           ; nope: go do another

evalEnd:MOVEM.L (SP)+, D0-D2/A0-A2; restore caller's registers
   RTS
   ;  Here when the resource forbids caching.
   ;  A2 points to the flag.
evalTurnOff:
   SF  (A2)                ; …disable it for faster call next time
   BRA.S  evalEnd          ; clean up and exit
   ; *** Impure *** flag: 0 means mapping disabled; non-zero means
enabled.
qtEnabled:
   DC.B   $FF,00           ; initially enabled; 2nd byte to align

   ;  Instructions and register lists to stick into OS routines.
   ;  Each is 2 words.
OSent: MOVEM.L OSregs, -(SP)
OSent2:   MOVEM.L OSregs2, -(SP)
OSexit: MOVEM.L (SP)+, OSregs
OSexit2:MOVEM.L (SP)+, OSregs2

; Toolbox trap replacement routines.  To be re-evaluated,
; these must be between qtToolStart and qtToolEnd.
; Nothing else must be in here - the evaluator
; walks through this as an array.
qtToolStart:  ; Beginning of Toolbox trap replacement routines.
   qtTool qtMoveTo,_MoveTo
   qtTool qtSetPort,_SetPort
            ; …add your own here…
qtToolEnd:    ; End of Toolbox trap replacement routines.
; OS trap replacement routines.  As with toolbox,
; keep only these in here.
qtOSStart:    ; Beginning of OS trap replacement routines.
   qtOS   qtHLock,_HLock
   qtOS   qtHUnlock,_HUnlock
            ; …add your own here…
qtOSEnd:      ; End of OS trap replacement routines.
   END
```

© The Definitive MacTutor, Vol. 4

Assembly Language Lab

FKEY that runs other FKEYs!

John Holder
Contributing Editor
MacTutor Vol. 4 No. 6

[John Holder is a previous contributor to MacTutor and is well known for his numerous FKEY shareware products. In this article, John shows us how to create an FKEY that runs other FKEYS without installing them in the system folder! Hence this FKEY has an event loop, menu bar and all the other goodies normally found in a regular application. The source code disk for this month also includes two of his more popular FKEYS to use as launch samples, along with a shareware application so you can contact John about other goodies he might have for you. -Ed]

What's an FKEY?

FKEY's (Function Keys) are resources containing executable code that are called upon by hitting the keys Command-Shift-# (any number 0-9) at the same time. In the standard System file there are two FKEY's; with an ID#3 and #4 that are used to dump the current screen or window onto disk or printer.

There are also many other FKEY's out in the Mac world, some Public Domain or Shareware, and others are sold commercially. They can be installed into the System file by using "ResEdit" or "Fkey Installer" (this application was created by "Dreams of the Phoenix, Inc™" and released into the public domain to encourage people to make Fkeys). The Fkey presented in this article can be used to access other Fkey's which are not in the System file!

What's this all about?

This article will show you how to create an FKEY with an interface similar to a standard applications (menus with command key equivalents, windows, and an event loop). It does not support DA's nor does it have an EDIT menu, but if you need to add this function to your own Fkey it would be very simple.

This Fkey was created with the Macintosh 68000 Development System. All applications are on a disk titled 'MDS1' and all .D files are on a disk titled 'MDS2'. First assemble the file 'FkeyTemp.Asm', then run the linker on the file 'FkeyTemp.Link', and finally run RMaker on the file 'FkeyTemp.R'. You will now have an FKEY ready to put into your System file!

Fkey Sampler, How it Works

First, we create an offset table that is used to store the FKEY's many "global" variables. This is done by naming the offsets that it will need such as sfReply, WindPtr, etc., and calculating how many bytes each will take (explained under "How to set up the offset Table").

When the Fkey starts, there is a short jump over its header, then the applications current port and all registers are saved, next it brings up the standard arrow cursor with _InitCursor. We now

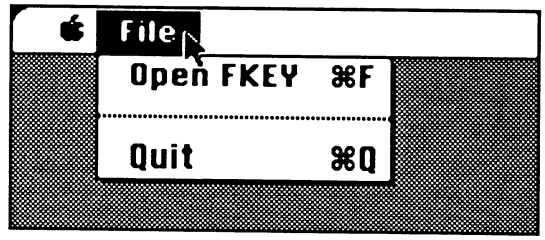


Fig. 1 Our FKEY launches other fkeys!

check the global variable MenuList to see if the application which is currently running has a menu, if it doesn't have one (like the MiniFinder) then we can't let the Fkey run! If it does have a menu, we try to allocate enough space for the Fkey's global variables by making a non-relocatable block with _NewPtr. If there is not enough memory for the globals (there has been an error if _NewPtr returns a zero in register A0) it beeps the speaker and returns to the Application. Otherwise, a pointer to the storage area is put into register A4.

If the storage was allocated, we go on and get a handle to a copy of the applications menu bar with _GetMenuBar, and clear the menu so that we may replace it with our own. After the Fkey's menus are created with _NewMenu, _AppendMenu, and _InsertMenu, we create a window that will simulate the standard desktop. Creating a new window generates an update event, so when it gets to the main event loop, the window will be filled with the current desktop pattern.

The Main Event

Now its off to the main loop to look for events. This Fkey uses only keydown, mousedown & update events. Upon a useable event we jump to the appropriate routine to handle it, if it's an update event the background window is filled with the current desktop pattern (found in the system global variable

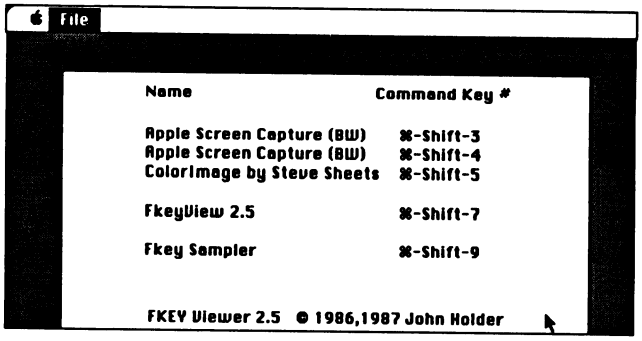


Fig. 2 A useful FKEY running under our FKEY

"DeskPattern"). Other events are handled just like a standard application by using a jump table to jump to the appropriate routines.

Doing the About...

When the mouse is clicked in the "About Fkey Sampler..." menu item under the Apple menu, a window is created and set as the current port. Then the windows text font and size is set to 9 point Monaco with _TextFont and _TextSize. Now the stack is set up for _TextBox (a very handy routine that displays text in a given rectangle with left, right or center justification!) Then we jump to a routine that uses _GetOSEvent to watch for a mousedown or key down event. When a key or the mouse button has been clicked we dispose of the window, unhighlight the menu and return to the main loop to get the next event!

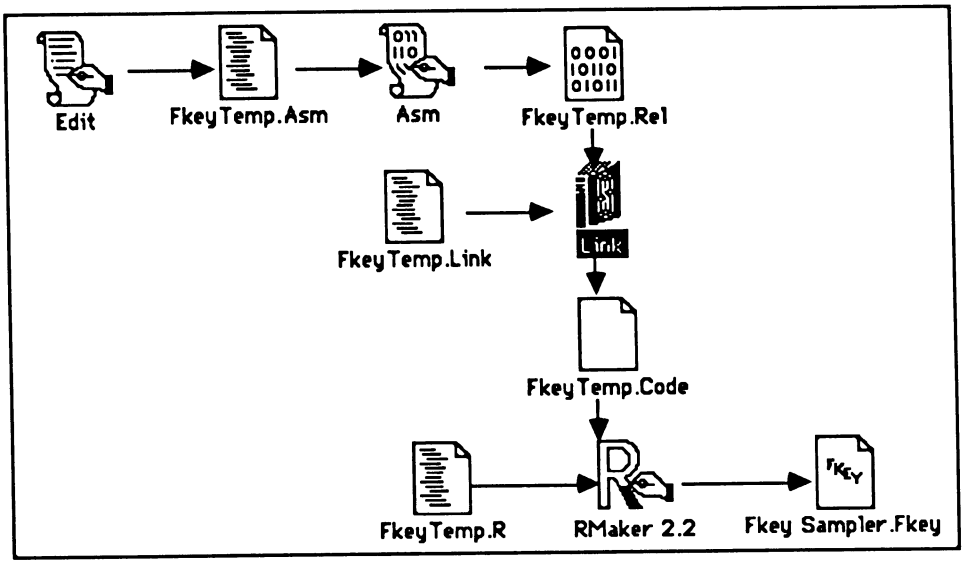


Fig. 3 Map for how the FKEY is constructed
Use ResEdit to copy the FKEY into the system file

Opening the Fkey

The purpose of this particular Fkey is to open and use other FKEY's (of Type 'FKEY') that are not in the System file. When this routine is called, the standard get file dialog is displayed with _Pack3. If a name is chosen, the volume the file is on is set as the current volume with _SetVol. Now we try to open the resource fork of the file (_OpenResFile will return a #-1 if it can't open a files resource fork). If the resource file opened with no problems we want to get the Fkey resource which is inside this file and execute it (run it, not kill it!).

Now we want to get the first Fkey resource in memory by using _GetIndResource (the newly opened res file is searched first). We check to make sure the returned Fkey is from the newly opened resource file by using _HomeResFile (when given a handle to a resource, this routine returns its resource files reference number). If it is from the Fkey file just opened, the Fkey resource is locked in memory (we don't want the Fkey moving!), the handle is made into a pointer and the Fkey is executed by jumping to it (with *JSR*). When that Fkey is done, we close the resource file and return for the next event.

Doing the Quit!

When an Fkey quits it must dispose of any memory it created. The first thing dumped is the window used as the desktop, then the menus it made are released from memory with _DisposMenu and the menu bar is cleared. Now the applications menu bar is restored, the memory we allocated for the globals is released, all events are flushed, the applications current port and registers are restored, and a return from subroutine *RTS* returns command to the application!

Making your own FKEY

All Fkey's must start with the FKEY header shown in the code listing. The only thing you will need to change in the header for your own Fkey is the ID number. You can use this Fkey to test all of your own Fkeys without installing them into the System file!

To make your own Fkey using this one as a template should be very simple. Just make your own offset table, or use this one and add your own needed offsets (explained below), then add your own menus and the appropriate routines to handle them. Also, change the contents of the text between MyTextBegin and MyTextEnd to whatever you want for your Fkeys 'About..' window.

A complete jump table for events and a jump table for where the mouse was clicked (returned by _FindWindow) is included to help in making your own Fkeys. Just remember to always check if there is enough memory before loading a resource or creating storage and to dispose of any storage that the Fkey created before returning control to the application!

How to set up the Offset Table

Lets say you need room for one handle (4 bytes) and two separate words (2 bytes each) of storage.

For the this storage you would need to put #8 into register D0 before calling _NewPtr (8 bytes are needed)

```
;Your Offset table (example)

AHandle  equ  0   ;4 bytes
AWord    equ  4   ;2 bytes
NextWord equ  6   ;2 bytes
```

Register A4 points to your global storage area . To store a word into the global "AWord" just use:

```
move  #99, AWord(A4)
```


© The Definitive MacTutor, Vol. 4

To put the value contained in AWord(A4) into Register D2 , use:

```
move  AWord(A4), D2
```

That's all folks...

Well, that's it, I hope you've learned something, and I hope to see lots of new Fkeys around!

```
;File:   FkeyTemp.Asm
;-----------------------
;An Fkey sampler Fkey
;By John Holder    2/8/87
;Revised    3/29/88
;-----------------------
    Include  Traps.D
    Include  Toolequ.D
    Include  Sysequ.D

;-----------------------
;Macros to save & restore the registers
;-----------------------
MACRO    SaveRegs =
    movem.l  A0-A4/D0-D7,-(SP)
    |

MACRO    RestoreRegs =
    movem.l  (SP)+,A0-A4/D0-D7
    |

;-----------------------
;All equates go here
;-----------------------
;these are offsets into the global storage.
;(a non-relocatable block we created pointed to by A4)
;-----------------------
sfReply      equ  0    ;sfreply record
GetVRef      equ  6
GetVers      equ  8
GFileName    equ  10
IOParamBlk   equ  76   ;IOparamblock (80 bytes)
WindPtr      equ  156  ;for our background
                       ;window
EventBlk     equ  160  ;Event Block
what         equ  160  ;Event number
message      equ  162  ;Additional information
when         equ  166  ;Time event was posted
where        equ  170  ;Mouse coordinates
modify       equ  174  ;State of keys and button
wWindow      equ  176  ;Window pointer
Menu1Hndl    equ  180  ;handle to our Apple menu
Menu2Hndl    equ  184  ;handle to our File menu
oldMenuBar   equ  188  ;handle to applications
                       ;menu
TheWindow    equ  192  ;for _FindWindow

;event masks for _GetOSEvent
mDownMask    equ  2
keyDownMask  equ  8

;Menu equates
AppleMenu    equ  1    ;Apple Menu
AboutItem    equ  1
FileMenu     equ  2    ;File Menu
OpenFkeyItem equ  1
QuitItem     equ  3

;-----------------------
```

```
;All FKEY's must have this FKEY resource header!
;-----------------------
    bra.s    StartIt      ;branch over header
    dc       0            ;flags (not used)
    dc.b     'FKEY'       ;resource type
    dc       9            ;Resource ID #
    dc       0            ;version # (not used)
;-----------------------
StartIt
    SaveRegs              ;save regs MACRO
    subq     #4,SP
    move.l   SP,-(SP)
    _GetPort              ;save Port for later
    _InitCursor           ;standard arrow cursor

    ;does application have a menu???  if not, then quit!
    move.l   MenuList,D0
    beq      NoMenuJump   ;no menu, beep & quit!

    move.l   #200,D0      ;200 bytes of storage
    _NewPtr,CLEAR         ;for our globals
    move.l   A0,A4        ;pointer in A4
    cmp.l    #0,A4
    bne.s    StorageOk    ;if not zero, ptr is ok!

NoMenuJump
    bsr      Mem_Error    ;otherwise beep
    bra      No_Mem       ;and quit

StorageOk
    bsr      SetUpMenus   ;set up our menus
    bsr      SetUpWindow  ;set up background
                          ;window
;-----------------------
MainLoop                          ;standard event loop
    _SystemTask
    clr    -(SP)
    move #$0FFF,-(SP)             ;event mask
    pea      EventBlk(A4)         ;Place to return results
    _GetNextEvent                 ;Look for an event
    move     (SP)+,D2
    beq      MainLoop             ;zero if we shouldn't
    move     What(A4),D0          ;respond
    add      D0,D0
    move     EventTable(D0),D0
    jmp      EventTable(D0)       ;jump to appropriate
                                  ;routine

EventTable
    dc   MainLoop-EventTable      ;null event
    dc   MouseDown-EventTable     ;mouse down
    dc   MainLoop-EventTable      ;mouse up
    dc   KeyDown-EventTable       ;key down
    dc   MainLoop-EventTable      ;key up
    dc   KeyDown-EventTable       ;auto key
    dc   UpDateJump-EventTable    ;update event
    dc   MainLoop-EventTable      ;disk event
    dc   MainLoop-EventTable      ;activate event
    dc   MainLoop-EventTable      ;abort
    dc   MainLoop-EventTable      ;network event
    dc   MainLoop-EventTable      ;I/O driver event

;-----------------------
SetUpWindow
    clr.l    -(SP)                ;for returned pointer
    clr.l    -(SP)                ;let Mac make storage
    pea      WindowRect           ;windows rectangle
    clr.l    -(SP)                ;no title
    move.b   #1,-(SP)             ;it is visible
    move     #plainDBox,-(SP)     ;Type of window
    move.l   #-1,-(SP)            ;in front
    move.b   #0,-(SP)             ;no close box
    clr.l    -(SP)                ;refcon zero
    _NewWindow                    ;make window!
```

```
    move.l   (SP),WindPtr(a4)  ;pointer into global
    _SetPort                   ;set as the port
    rts

;_____________________
SetUpMenus
    clr.l    -(SP)             ;get applications menu
    _GetMenuBar                ;bar handle for later
    move.l   (SP)+,oldMenuBar(A4)  ;restoration!
    _ClearMenuBar              ;clear Apps menu bar

;make all of our menus
;the apple menu
    clr.l    -(SP)
    move     #1,-(SP)
    pea      AppleSymbol
    _NewMenu
    move.l   (SP)+,Menu1Hndl(A4)
    move.l   Menu1Hndl(A4),-(sp)
    pea      'About Fkey Sampler...'
    _AppendMenu
    move.l   Menu1Hndl(A4),-(sp)
    move     #2,-(SP)
    _InsertMenu
;the File menu
    clr.l    -(SP)
    move     #2,-(SP)
    pea      'File'
    _NewMenu
    move.l   (SP)+,Menu2Hndl(A4)
    move.l   Menu2Hndl(A4),-(sp)
    pea      'Open FKEY/F'
    _AppendMenu
    move.l   Menu2Hndl(A4),-(sp)
    pea      '(-'
    _AppendMenu
    move.l   Menu2Hndl(A4),-(sp)
    pea      'Quit/Q'
    _AppendMenu
    move.l   Menu2Hndl(A4),-(sp)
    clr      -(SP)
    _InsertMenu
    _DrawMenuBar       ;draw the new menu bar
    rts

UpDateJump
    bsr      UpDateWindow
    bra      MainLoop

;_____________________
;This routine keeps our background window (The DeskTop!)
;the same pattern as the current standard desktop!
;_____________________
UpDateWindow
    move.l   WindPtr(A4),-(SP)  ;our background
                                ;window
    _BeginUpdate                ;is kept in WindPtr(A4)
    move.l   WindPtr(A4),-(SP)
    _SetPort
    pea      PaintRect     ;the rectangle to paint
    pea      DeskPattern   ;global var, contains
    _FillRect              ;pattern used for desktop
    move.l   WindPtr(A4),-(SP)
    _EndUpdate
    rts

;_____________________
;A key down event occured, check if the command key was
;down, if not, get the next event!
;_____________________
KeyDown
    move     Modify(A4),D1
    btst     #8,D1             ;check command key bit
    beq      MainLoop          ;zero if command key
                               ;is not down!
    clr.l    -(sp)
    move     message+2(A4),-(sp)   ;put char onto stack
    _MenuKey                   ; for _MenuKey
    move.l   (sp)+,D4          ;put result in D4
    beq      MainLoop          ;if zero, no menu equivalent
    bra      WhatMenu

;_____________________
;A mousedown event occured, we only check if its in a menu, if
;not then get the next event!
;_____________________
MouseDown
    clr      -(SP)             ;word result
    move.l   where(A4),-(SP)   ;where field of
                               ;event record
    pea      TheWindow(A4)     ;returns the window
                               ;pntr the
    _FindWindow                ;mouse was clicked in,
    move     (SP)+,D0          ;if it was in one.

    add      D0,D0
    move     WindowTable(D0),D0
    jmp      WindowTable(D0)   ;jump to appropriate
                               ;routine

;We only use InMenu event!
WindowTable
    dc   MainLoop-WindowTable  ;In Desk
    dc   InMenu-WindowTable    ;In Menu Bar
    dc   MainLoop-WindowTable  ;In System Window
    dc   MainLoop-WindowTable  ;in Content
    dc   MainLoop-WindowTable  ;in drag
    dc   MainLoop-WindowTable  ;in grow
    dc   MainLoop-WindowTable  ;in go away

;_____________________
;The mouse was clicked in the menu bar.
;_____________________
InMenu
    clr.l    -(SP)
    move.l   where(A4),-(SP)
    _MenuSelect
    move.l   (SP)+,D4
    beq MainLoop               ;zero if no item selected

;jump to here after command key down event
WhatMenu
    move.l   D4,D6             ;D6 will now have the
    swap     D4                ; item # D4 will have menu #
WhichMenuWasIt
    cmp      #AppleMenu,D4     ;is it in the apple menu?
    beq      InAppleMenu       ;if so, go see which item!
    cmp      #FileMenu,D4      ;is it in the File menu?
    beq      InFileMenu
    bra      MainLoop          ;where else could it be!

InAppleMenu                    ;in the "Apple" menu
    cmp      #AboutItem,D6
    beq      DoAbout           ;its in the About... Item!
    bra      MainLoop

InFileMenu                     ;in the "File" menu
    cmp      #OpenFkeyItem,D6  ;in the OpenFkey Item?
    beq      Open_Fkey         ;if so go open it!
    cmp      #QuitItem,D6      ;Quit item?
    beq      QuitRoutine       ;if so go quit
    bra      MainLoop

;_____________________
;Show the "About Fkey Sampler..." window
;_____________________
DoAbout
    saveregs                   ;save the registers
```

```
    clr.l   -(SP)
    clr.l   -(SP)
    pea     AboutRect
    move.l  #0,-(SP)
    move.b  #1,-(SP)
    move    #1,-(SP)
    move.l  #-1,-(SP)
    move.b  #0,-(SP)
    clr.l   -(SP)
    _NewWindow                  ;make a window
    move.l  (SP),A2             ;put pointer into A2
    _SetPort                    ;make it the port

    move    #Monaco,-(SP)       ;Use the Monaco Font
    _TextFont
    move    #9,-(SP)        ;make it 9 point (size)
    _TextSize

    pea     MyTextBegin   ;address of "About" Text


;calculate how many chars in text for TextBox
    move.l  #MyTextEnd-MyTextBegin,-(SP)
    pea     TextRect            ;rectangle for text
    move    #1,-(SP)            ;center justification
    _TextBox
    bsr     Wait_For_Event      ;wait for button/key down
    move.l  A2,-(SP)
    _DisposWindow               ;get rid of the window
    bsr     UnHiliteIt          ;go unhilight the menu!
    move.l  WindPtr(A4),-(sp)
    _SetPort                    ;reset our old port
    restoreregs                 ;restore registers
    bra     MainLoop            ;go get another event

;-------------------------
;Wait for mouse or key down event
;-------------------------
Wait_For_Event
    link    A6,#-evtBlkSize         ;link, event block size
WaitLoop
    lea     -evtBlkSize(A6),A0      ;ptr to event block in A0
    moveq   #mDownMask!KeyDownMask,D0 ;Event masks in D0
    _GetOSEvent
    beq     Clicked             ;if zero, time to leave!
    bra     WaitLoop            ;loop until event
Clicked                         ;mouse or key clicked!
    unlk    A6                  ;unlink A6
    rts

;-------------------------
;Quitting, dispose of all our storage and menus and return to
;the application that the Fkey was called from!
;-------------------------
QuitRoutine
    move.l  WindPtr(A4),-(SP)       ;dump background
    _DisposWindow                   ;window
    move.l  Menu1Hndl(A4),-(SP)     ;dump our #1
    _DisposMenu             ;menu
    move.l  Menu2Hndl(A4),-(SP)     ;dump our #2
    _DisposMenu                     ;menu
    _ClearMenuBar                   ;clear the menu
    move.l  oldMenuBar(A4),-(SP)
    _SetMenuBar                     ;restore apps menu bar!
    move.l  oldMenuBar(A4),A0       ;dump copy of
    _DisposHandle                   ;apps menu handle!
    _DrawMenuBar                    ;draw it
    move.l  A4,A0
    _DisposPtr                      ;dump our global
                                    ;storage

No_Mem
    move.l  #$0000FFFF,D0       ;flush all events
    _FlushEvents
    _SetPort                        ;set original port!
    restoreregs                     ;restore the registers
    rts                             ;return to the applica-
tion!

;-------------------------
;Couldn't allocate space for our global storage area
;just beep & quit!
;-------------------------
Mem_Error
    move    #10,-(SP)
    _SysBeep
    rts

;-------------------------
;Unhilight menu bar after mouse down in a menu item
;-------------------------
UnHiliteIt
    move    #0,-(SP)
    _HiliteMenu
    rts

;-------------------------
;Get a file of type "FKEY", open its resource fork, and
;execute the Fkey (if there is one) contained in it!
;-------------------------
Open_Fkey
    saveregs                    ;save registers
    move    #82,-(SP)           ;coordinates for GetFile
    move    #100,-(SP)          ;dialog box
    clr.l   -(SP)               ;prompt
    clr.l   -(SP)               ;File Filter Proc Pointer
    move    #1,-(SP)            ;number of file types
    pea     TypeList            ;our type list
    clr.l   -(SP)               ;Dialog Hook Proc Pointer
    pea     sfReply(A4)         ;sfreply record
    move    #2,-(SP)            ;#2 for SFGetFile
    _Pack3                      ;Call the Get File Pack
    cmp.b   #0,sfReply(A4)
    beq     Cancel              ;if user hit cancel, quit!

    bsr     UpDateWindow        ;update window before
                                ;opening the Fkey
    lea     IOParamBlk(A4),A2   ;the IO param block
    clr.l   12(A2)              ;zero "ioCompletion"
    lea     GFileName(A4),A0    ;put a pointer to the files
    move.l  A0,18(A2)           ;name into "ioNamePtr"
    move    GetVRef(A4),22(A2)      ;"ioVRefNum"
    move.l  A2,A0               ;put param block ptr in A0
    _SetVol                     ;set as current volume

    clr     -(SP)               ;space for refnum
    pea     GFileName(A4)       ;name of res file to open
    _OpenResFile                ;Open that resource file
    move    (SP)+,D7            ;put result in D7
    cmp     #-1,D7              ;will be -1 if any
                                ;errors!
    beq Cancel                  ;if error go to cancel

;this stops resources from being loaded with GetIndResource,
;GetResource or GetNamedResource
    move.w  #0,-(sp)
    _SetResLoad

    clr.l   -(sp)
    move.l  #'FKEY',-(sp)
    move    #1,-(sp)
    _GetIndResource             ;get first FKEY resource
    move.l  (sp)+,A3            ;handle into A3

;Turns Automatic resource loading back on!
    move.b  #1,-(sp)
    _SetResLoad
```

```
    clr.l   -(sp)
    move.l  A3,-(sp)
    _SizeRsrc
    move.l  (sp)+,D2          ;how big is the resource?
    cmp.l   #-1,D2            ;-1 = no resource
    beq     NotEnoughMem      ;no resource error so quit

;do we have enough memory to load the FKEY
    move.l  D2,D0
    _NewHandle
    move.l  A0,A1
    cmp.l   #0,A1             ;was zero returned?
    beq     NotEnoughMem      ;not enough room, so quit!
    _DisposHandle             ;otherwise dispose the new
                              ;handle, and go on..
    move.l  A3,-(sp)
    _LoadResource             ;Load resource into memory!
    clr     -(sp)
    move.l  A3,-(SP)
    _HomeResFile              ;what resource file
    move    (sp)+,D2          ;is this resource from?

    cmp     D2,D7             ;is it from new file?
    beq     YepItsFromNewFile ;yep!
    bra     NotEnoughMem      ;if not, close the res
                              ;file and quit!
YepItsFromNewFile
    move.l  A3,A0
    _Hlock                    ;Lock it before jumping to it
    move.l  (A3),A2           ;put pointer to FKEY in A2
    SaveRegs                  ;save the regs
    jsr     (A2)              ;Run the Fkey!!
    RestoreRegs               ;restore the regs
    move.l  A3,A0
    _HUnlock                  ;UnLock FKEY handle
    bsr     CloseIt           ;go close res file

Cancel
    restoreregs               ;restore the regs
    bsr     UnHiliteIt        ;unhilight the menu
    bra     MainLoop          ;get the next event


NotEnoughMem
    bsr     CloseIt
    bra     Cancel

CloseIt
    move    D7,-(SP)
    _CloseResFile             ;Close the res file
    rts

;-------------------
;All constants go here
;-------------------
;rect of background (desktop) window
;make it giant to cover entire background!

WindowRect  dc  0,0,4000,4000
PaintRect   dc  0,0,4000,4000
TypeList    dc.b 'FKEY'

    .ALIGN  2

;rect of "About..." window
AboutRect   dc  130,100,190,412

;rect text will be displayed in for _TextBox
TextRect    dc  10,10,90,302

AppleSymbol dc.b 1,$14    ;apple symbol for menu #1
    .ALIGN  2

MyTextBegin dc.b 'Fkey Sampler 1.0',$0D,$0D
            dc.b 'Written by John Holder'
    .ALIGN  2
MyTextEnd

    END
```

```
;The Linker file     (file name FkeyTemp.Link)

]

FKEYTemp
/Output FkeyTemp.Code
/Type 'TEMP'
$
```

```
*The RMaker file   (file name FkeyTemp.R)

Fkey Sampler.Fkey
FKEYQD15

TYPE FKEY = PROC
Fkey Sampler,9
FkeyTemp.Code
```

Assembly Language Lab

Font Dialog Box Using List Manager

Ray A. Cameron
Brunswick, Victoria
Australia
MacTutor Vol. 4 No. 7

Ray. A. Cameron is an Electrical Engineer from Brunswick, Victoria in Australia, employed by Telecom, the Australian Telecommunication Commission. He is currently involved in the introduction of optical fibre into the Customer Access Network. This is his first article for MacTutor.

MacTutor Down Under

Before I start writing about my routines I'd like to say how fantastic MacTutor is for the service that it is providing to Mac programmers in disseminating programming ideas and techniques. It doesn't matter which language we program in, it is possible to gain information from every article published in MacTutor.

Something that I know would be beneficial to all readers of MacTutor is if the writers of articles include in their description not only how the program works (as published) but what problems they have encountered along the way or techniques thay have employed and why. Also any idiosyncrasies in the system software that don't appear to be fully explained in Inside Macintosh.

What This Program Does

The FontDialog routine displays a Modal Dialog box for the user to select a font name, size and style. The routine is designed to be incorporated into a program, as I have done with the "Window" example (distributed with the old MDS software and included here), which is on this month's source disk. There are two routines that I have written, the first routine (SetupFontMap) creates a record of all the 'FONT' resources available to the user in the system file. It doesn't look for 'FOND' or 'NFNT' resources or fonts attached to documents. The next routine (FontDialog) creates a Modal Dialog box (Fig 1) so that the user can choose a font name, size and style. The font name and size lists are created with the List Manager from data stored in the FontMap record. The routine also includes a UserFilter with the Modal Dialog to handle events.

Some of the procedures and functions used in the FontDialog routine take up a number of lines and are used often so I've converted them into Macros to save space. I won't describe the Macro file as it should be self explanatory.

User Interface

The Dialog box presented in this example is similar to the one in Microsoft Word. Now that we can all use nested menus to select font, size and style, you might want to be old fashioned and go back to a text dialog box instead! You can select a font name

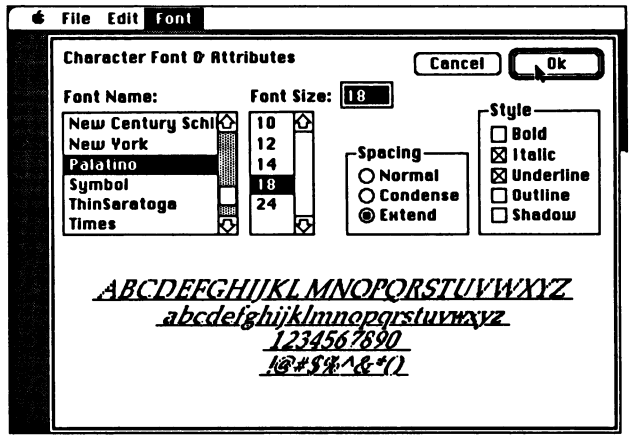


Fig. 1 Our Font Dialog Box selects text, size & style

by either clicking on a font name or by typing a character to select the first name starting with that character (or the first name to follow in alphabetical order). As you continue to type additional characters (up to 31) any file that matches the characters you type is found and selected. If you pause while typing, the next character is considered to be a new request, rather than a continuation of your first request. The Delay Until Repeat setting which you set in the (Control Panel) determines how long you can pause before additional characters are considered to be a new request. Hence our dialog box is fairly intelligent in that it includes a sort and search routine and uses the list manager, useful techniques for a variety of dialog functions.

The Font Size Selection Window displays the sizes that are currently available on the system (for the selected font name). If no font name is selected then the Font Size Selection Window is

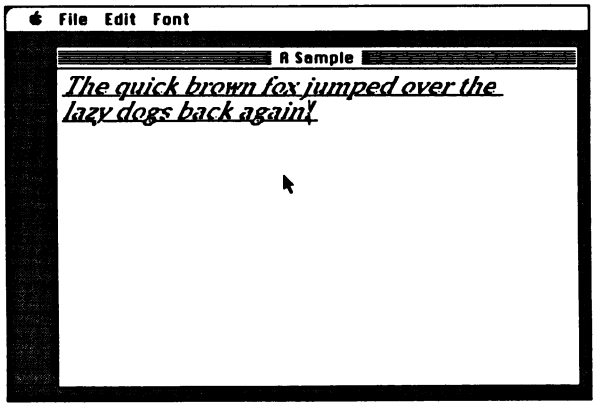
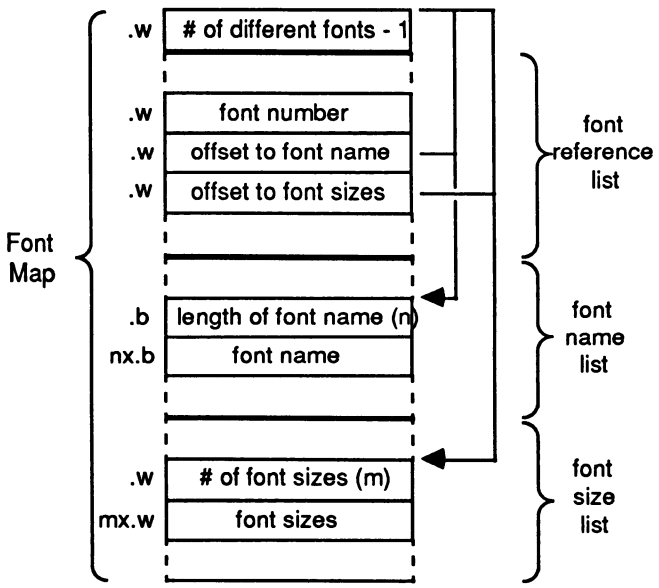


Fig. 2 After text selection, our sample text window allows typing in the chosen font & style



Format of theFont Map

Fig. 3 Layout of our custom font map structure

blank, this doesn't affect the font size in the textedit window. A font size can be selected by clicking on the desired size or by typing in the font size. Numbers and the 'back space' character are assumed to be an input to alter the font size, whilst all other characters (except 'cr' & 'enter') are used to find a matching font name. The font style has been separated into two sections, 'spacing' (condense and extend) and 'style' (outline, bold, etc). When the dialog box is first displayed the radio buttons and check boxes reflect the current settings. If their settings are altered it is possible to reset them to their original values by clicking on the group title concerned (either 'spacing' or 'style'). The settings are only reset if the mouse is released inside the title. To indicate that the mouse is inside the title the thickness of the frame around the group doubles. This type of facility (interface) lends itself to having the groups set to normal or plain if a double click occurs in the title, something which I haven't implemented here.

At the bottom of the dialog box is displayed some (sample) text showing the affect of the selected settings. The sample text will only be displayed if the current settings of font name and size are valid. A valid font size can be from 4 to 127, normal quickdraw limitations.

If the Ok button is selected or the 'cr' or 'enter' keys are pressed while the current settings are invalid an alert box is displayed indicating which setting isn't valid.

FontMap

To produce the dialog box I require the names of the available fonts on the system as well as the sizes available for each font. The idea of a font menu with a range of font sizes on it (ie MacWrite) is relatively easy, all that is required is to test if the selected font has a desired size. If it does then the size is shown in outlined style, whilst if it doesn't its style is plain. To find every size available on the system for a particular font name is a little harder. I decided to produce a relocatable block which would contain the names of all the fonts (in alphabetical order) and the sizes of each font (in numerical order). Doing this reduces the amount of time required to set-up the Font Name and Font Size Selection Windows. Note that even some commercial software doesn't bother to alphabetize font lists! A graphical representation of the FontMap is shown in Fig 3. The first (word) field contains the number of different fonts minus one. Then there are font references, one reference (of three words) for each font. The first field is the font number, the next is the offset to the font name and the last is the offset to the font size list. Both offsets are from the start of the FontMap. The font references are sorted into the alphabetical order of the font names. The space allocated for each font name is the smallest number of even bytes that the name will fit in. The font sizes for each font are sorted into their numerical order within each font size list. The first word of the font size list is the number of font sizes in the list, then follows the font sizes, one word per font size. The handle to the Map is stored for access by the FontDialog routine and could be used to prepare the font sizes in a menu. By having the names and sizes sorted within FontMap it means that when they are placed into a list their cell position relates to their position in the FontMap.

Apple have laid down in their user interface guidelines that font sizes that exist on a system be represented in some way, as in outlining in a menu etc. With the new font family resources ('FOND's & 'NFNT's) it is possible to have a font that already has a style attribute associated with it, like Geneva Shadow. How are these fonts to be represented so that the user can tell them apart from a plain font size? The text sample is one solution and perhaps there are others.

Creating the Font Map

The SetupFontMap routine is called at the start of a program so that the FontMap can be used to set up the font size menu items. The routine, first of all evaluates what size to make the Map (relocatable block). It examines each 'FONT' resource on the system and determines if the resource contains a font name or a font size. If the resource is a font name, the font (resource) ID and the pointer to the name are stored on the stack. If the resource is a font size, the font (resource) ID is stored in a non-relocatable block (SizeBlock). As each resource is examined, a running total is kept of how large to make the FontMap and how many different fonts there are. The FontMap is created and the number of font names minus one is loaded into the first field. The font ID's and name pointers are loaded into the font references from the stack. The font references are then sorted into the alphabetical order of the font names. The font names are then loaded into the Map, disposing of their original non-reloctable block in the process and replacing the name pointer with an offset. SetupFontMap then goes through SizeBlock and loads in all the font sizes associated with the first font. The sizes are then sorted into

numerical order. SetupFontMap then continues until all fonts have their sizes loaded and sorted. All temporary blocks are disposed of as the routine is finished with them. The FontMap is then unlocked and control returns to the main program.

This routine should be rewritten so that it examines the font resources ('FOND ', 'FONT' and 'NFNT') and the font resources associated with a document. It should also be recalled each time a new document is accessed in case there is a new font resource associated with it. But that's a task I'll leave for you. *[Note that Apple discourages the use of fonts within documents now. -Ed]*

Font Dialog Mechanics

The FontDialog routine would generally be accessed by the selection of a menu item. The items associated with a dialog are stored in an item list which contains the item type, whether it's enabled or not and a pointer to the routine that will draw the item, as well as other variables. The order of the items in the list is very important (as I found out). The items are drawn and accessed in the order that they appear in the item list. If items overlap, the item highest in the list (lowest item number) takes precedent. I wanted to set up the 'spacing' and 'style' groups as shown in Fig 1 so that if I clicked on either the title or a control, the program returned from the ModalDialog trap. At this stage of writing the program I wasn't using a UserFilter so I had to create routines to draw the bold frame around the Ok button and thin frames around the control groups. The problem arose when I wanted to draw the frame and the group title so that the frame wasn't drawn through the title (as per Fig 1). If the frame is drawn then the title, the routine that draws the title erases a section of the frame where the title goes. To do this the useritem would have to have a lower item number than the title. The useritem that represented the frame was disabled so I thought that the other items that were enabled wouldn't be affected by it. Wrong! By placing the useritem higher in the item list than the other items I had made all the lower items except for the top half of the title effectively invisible/disabled. To get around this problem (before I used a UserFilter) I placed the useritem lower in the item list and wrote a subroutine similar to TrackHeading. The routine created a region consisting of the useritem rect minus the group title rect, and made it the clipping region, it then drew a frame around the useritem rect and reset the clipping region. For details on regions and clipping check Chris Derossi's article on "Quickdraw Does Regions!" (Pascal Procedures) in MacTutor Vol 1 No 3, February 1985. When a User Filter is used, the programmer isn't confined to accessing the dialog items in their order, which simplifies the drawing of the items.

The FontDialog routine creates a non-relocatable block into which it stores all its variables. This was done so that memory is allocated only when the routine is called. It allows the main program to check to see if there is enough spare memory before the routine is called, or the routine could become a separate code segment that isn't loaded into memory until it's needed. The routine takes copies of the font number, size and face (style) that are passed to it so that these can be altered without affecting the original values, in case the user decides to cancel the dialog. The routine only accepts a single font variation (number, size and style), it isn't suitable for altering multiple font variations.

The spacing radio buttons and style check boxes are set up to display the original settings. I made the condense and extend style variations radio buttons because there is no point in both variations being selected (they cancel one another out).

The Font Name and Size Selection Windows are then created and the available font names and sizes are loaded into their respective list. Each list has its selection flags set so that only one cell is selected at a time and empty cells aren't selectable. Even though the selection flag was set so that only one cell is selected, I found that I could select more than one cell by using the LSetSelect trap. I wanted to set a cell and I assumed that the List Manager would deselect the currently selected cell. Instead I had to deselect the currently selected cell and then select the desired cell (as shown in DKeyDown). When the mouse is used to select a cell, only one cell is selected, all others are deselected.

Only enough cells are created in each list to contain the font names and sizes. In some cases there are blank lines in the list. Clicking below the cells deselects the current selection, sometimes. For some reason it is possible to click below the cells in one spot and have the selection deselected while not in others. The area in which this occurs is the height of a cell.

A Sample Display Text Edit Record is then setup so that the 'STR ' resource number 264 is displayed in the useritem rect at the bottom of the dialog box. The string is displayed with the current font settings, if they are valid, if they aren't valid then no text is displayed. The text is centered horizontally within the useritem rect, and also vertically if all the lines are visible.

The handles of the string resources to be used by the alert window are then loaded into the local variable block.

As the user modifies the settings in the dialog box, the routine checks to make sure that they are valid. If they aren't valid a flag (OkActive) is cleared. I originally set the Ok button to be hilited with #254 (if the selections are invalid) which is supposed to make the control inactive but still indicate if it has been clicked in. When I did this, I found that the Ok button would be grey but it wouldn't indicate if it was clicked in. If I dragged the mouse an outline of the Ok button would move in the vertical axis. The problem occured when I let the default Modal Dialog routine handle the mouse or when I caught the event in the UserFilter and used the TrackControl trap. For this reason I've left the Ok button active when the settings are invalid and display an alert if the Ok button, 'cr' or 'enter' are selected. Before the alert is displayed the routine determines which variables aren't valid and sets up the param strings accordingly. I haven't checked to see if this still occurs since my Mac has been "enhanced". From a user interface perspective it is good practice to inform the user what the problem is instead of not accepting their input or sounding a beep.

After the items have all been set up the ModalDialog trap is called.

Dialog Filter Proc

The DUserFilter filters all the events that it needs to handle and lets the default routine handle the remainder (by returning false). DUserFilter handles its events and returns true and an item No. if the main program is to do some variable updating or house keeping due to the event (ie a new font name is selected). When the DUserFilter handles an event and a setting hasn't altered, ie a click on the selected font name, the event is converted to a null event.

DNull checks to see if the mouse is within the textedit rect, if it is it converts the cursor to a handle, otherwise it sets it to the arrow. This section of code could be sped up by storing the textedit item rect in the local variable block instead of using the GetDItem trap every time. When ever a font attribute is altered the sample text is erased and the routine waits until KeyThresh ticks have occured and then redraws the sample with the new attributes. The DNull code checks to see if KeyTresh ticks have passed and if the sample requires updating. If it does then the DUserFilter returns true and item number 21 (a disabled useritem).

DMouseDown handles the mouse down events within the two group titles and the two list windows. All other mouse down events are handled by the default ModalDialog routines. When a mouse down occurs in either of the two group titles the routine (TrackHeading) tracks the mouse to see if it is released in it or not. If it is, ModalDialog returns with the item number, otherwise the event is set to null. When a mouse down occurs in either of the List Manager windows the routines check to see if the selection has altered. If it has ModalDialog returns with the item number, otherwise the event is set to null.

DKeyDown checks for a 'cr' or 'enter' key and returns item No 1 (Ok button) if it finds one. It then checks to see if a number or 'back space' character has been pressed, if one has it deselects the selected cell in the Font Size Selection Window (if one is selected) and returns false for the default ModalDialog routine to handle it. DKeyDown then uses the character to search for a font name (NameSearch), as described in the User Interface section. The List Manager trap LSearch isn't suitable because it only checks for exact matches and I required the routine to check strings using the IUMagString trap. The FontMap is searched through because it is quicker than using the List Manager routines to check each cells contents. The routine then makes sure the desired cell is selected and visible.

DUpdate updates all the items in the dialog box. The update event removes the need to have an item number for every item that is drawn. A prime example is the bold box around the Ok button, another is the frame around the textedit item. Items don't have to be drawn in the order in which they appear in the dialog's item list. This allows the frames to be drawn around the 'spacing' and 'style' control item groups before the group titles are drawn. The programmer has complete control over what is drawn in the dialog and could in fact draw something that isn't related to an item in the item list. When lists are drawn they don't have a frame drawn around them, so I've had to include one on each list. After everything has been updated the event is set to a null event (SetoNull) and returned.

DActivate activates the dialog box (the two list windows and the dialog box's text edit record). I've also included the code to deactivate the dialog box, even though it never gets used. When the ShowWindow trap is called to display the dialog box the window is created, a deactivate event is created for the original front window as is an activate event for the dialog window. The main program's deactivate code isn't accessed because its event loop has been broken, the main program has to deactivate the front window before the dialog box routine is called. DActivate has to be able to filter out the (de)activate event for the original front window and not handle it. When the alert is shown, the dialog should be deactivated (just like the case above) with similar code to that in DDeactivate. The LActivate trap removes (hides) scroll bars when the list is deactivated instead of inactivating them so I didn't include the deactivate code because it was visually undesirable (sloppy).

The routines that handle the result of the ModalDialog trap perform the following functions:

DOk displays an alert if the settings aren't valid and returns to the ModalDialog loop (WaitOk). If the settings are valid the original values are replaced with the desired values. The routine then passes onto DCancel.

DCancel closes the dialog box and disposes of the unwanted records and the local variable block. When there is a lot of information to be included (displayed) in a dialog, it is best to initially set the dialog to be invisible, setup the items involved and then show the dialog. This creates a clean crisp presentation, as opposed to items appearing slowly with delays between items. The same philosophy applies when the dialog is being closed. It is better to hide the dialog first and then to dispose of the items rather than to see items within the dialog (window) disappear before the window does, very sloppy.

Style resets the control check boxes to their original settings after the user has clicked in the style title. SStyle toggles a check box after it has been clicked in.

Spacing resets the control radio buttons to their original settings after the user clicked in the spacing title. SSpacing sets the radio button that was clicked in and clears the other radio buttons within the group.

The ModalDialog default routine handles a mouse down event in the control items (radio buttons and check boxes) and tracks the mouse and returns (with that items number) if the mouse is released within the item. The main routine then alters the value of the control, knowing that it was clicked in, this allows groups of radio buttons to be organised together.

FontName updates the font number variable after a new font name was selected. The Font Size Selection Window is then updated to show the sizes associated with the new font name. If no font name is selected the Font Size Selection Window is cleared.

FontSize updates the font size variable after a new font size was selected from the Font Size Selection Window. The textedit

item (No 20) is then updated to display the new font size. If no font size was selected the textedit item retains its previous value (it isn't set to zero).

GetEditSize updates the font size variable after the user has altered the textedit item through a key stroke (0 through to 9 or 'backspace'). The currently selected size in the Font Size Selection Window is deselected. The ModalDialog default routine returns with the textedit item number when the mouse is clicked within its rect even though its value doesn't change.

FontSample updates the font attributes associated with the sample text at the bottom of the dialog box. The text is then centered vertically within the user item rect if the height of the text lines is less than the height of the rect. The sample is then displayed with the new attributes.

Each of the above routines have code included in them to check if their font variable has in fact altered, if it has the sample text is then erased and isn't updated until KeyThresh ticks have lapsed.

I'll leave the remaining description of the routines to the comments placed within each listing. Best of luck with writing your dialog.

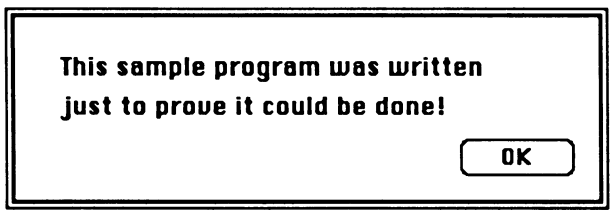


Fig. 4 A simple About Box Dialog

```
; File Window.Asm
;------------------------------------------------------------
;       Macintosh 68000 Development System
;------------------------------------------------------------
; The Window Sample Program issued with the MDS.
; Modified by Ray.A.Cameron to demonstate a Modal Dialog Box.
; Mon May 4, 1988 21:29:54

   This application displays a window within which you can
enter and edit text. Program control is through four menus:
the Apple menu, the File menu, the Edit menu, and the Font
menu.The Apple menu has the standard desk accessories and an
About feature. The File menu lets you quit the application.
The Edit menu lets you cut, copy, paste, and clear the text in
the window or in the desk accessories.  Undo is provided for
desk accesories only. Command key equivalents for undo, cut,
copy, and paste are provided. The Font menu lets you display a
Modal Dialog Box to select a Font Name, Size and attributes.

;---------------------- Includes ----------------

Include   Traps.D      ; Use System and ToolBox traps
Include   ToolEqu.D    ; Use ToolBox equates
Include   QuickEqu.D   ; Use QuickDraw equates
Include   SysEqu.D     ; Use System equates
Include   PackMacs.Txt ; Use Package equates

;------------------- Use of Registers -----------

; Operating Sys and Toolbox calls preserve D3-D7, A2-A4.

; Register use: A5-A7 are reserved by the system
;       D1-D3, A0-A1 are unused
;       D0 is used as a temp

ModifyReg     Equ    D4     ; modifier bits from GetNextEvent
MenuReg       Equ    D5     ; menu ID from MenuSelect,MenuKey
MenuItemReg   Equ    D6     ; item ID from MenuSelect,MenuKey
AppleHReg     Equ    D7     ; handle to the Apple Menu

TextHReg      Equ    A2     ; handle to TextEdit record
WindowPReg    Equ    A3     ; pointer to editing window
EditHReg      Equ    A4     ; handle to Edit menu

;------------------- Equates ----------------

; These are equates associated with the resources
; for the Window example.

AppleMenu Equ    1  ; First item in MENU resource
  AboutItem  Equ    1  ; First item in Apple menu

FileMenu      Equ    2  ; Second item in MENU resource
  QuitItem    Equ    1  ; First item in File menu

EditMenu      Equ    3  ; Third item in MENU resource
  UndoItem    Equ    1  ; Items in Edit menu
  CutItem     Equ    3  ; (Item 2 is a line)
  CopyItem    Equ    4
  PasteItem   Equ    5
  ClearItem   Equ    6

FontMenu      Equ    4  ; Fourth item in MENU resource
  DialogItem  Equ    1  ; Only item in Font menu

AboutDialog   Equ    1  ; About dialog is DLOG resource #1
ButtonItem    Equ    1  ; First item in DITL used by DLOG #1
ASample       Equ    1  ; Sample Window is WIND resource #1

;-------------------- Xdefs ----------------

; Xdef all labels to be symbolically displayed by debugger.

   Xdef      Start
   Xdef      InitManagers
   Xdef      SetupMenu
   Xdef      SetupWindow
   Xdef      SetupTextEdit
   Xdef      Activate
   Xdef      Deactivate
   Xdef      Update
   Xdef      KeyDown
   Xdef      MouseDown
   Xdef      SystemEvent
   Xdef      Content
   Xdef      Drag
   Xdef      InMenu
   Xdef      About

   XRef      SetupFontMap        ; Routines
   XRef      FontDialog

   XRef      FontMap             ; Variable

;----------------- Main Program -------------
Start
   Bsr       InitManagers        ; Initialize managers
   Bsr       SetupMenu           ; Build menus, draw menu bar
   Bsr       SetupWindow         ; Draw Editing Window
   Bsr       SetupTextEdit       ; Initialize TextEdit
   Bsr       SetupFontMap        ; Initialize the FontMap
EventLoop                        ; Main Program Loop
   _SystemTask                    ; Update Desk Accessories
   ; Procedure  TEIdle (hTE:TEHandle);
   Move.l       TextHReg,-(SP)      ; Get handle to text
```

```
record
   _TEIdle                          ; blink cursor etc.

   ; FunctionGetNextEvent(eventMask: Integer
   ;      Var theEvent: EventRecord) : Boolean
   Clr   -(SP)                  ; Clear space for result
   Move      #$0FFF,-(SP)       ; Allow 12 low events
   Pea       EventRecord        ; Place to return results
   _GetNextEvent                ; Look for an event
   Move      (SP)+,D0           ; Get result code
   Beq   EventLoop              ; No event... Keep waiting
   Bsr   HandleEvent            ; Go handle event
   Beq       EventLoop          ; Not Quit, keep going
   Rts                          ; Quit, exit to Finder

   Note: When an event handler finishes, it returns the Z flag
set.  If Quit was selected, it returns with the Z flag clear.
An Rts is guaranteed to close all files and launch the Finder.

;------------------- InitManagers -------------

InitManagers
   Pea          -4(A5)          ; Quickdraw's global area
   _InitGraf                    ; Init Quickdraw
   _InitFonts                   ; Init Font Manager
   Move.l       #$0000FFFF,D0   ; Flush all events
   _FlushEvents
   _InitWindows                 ; Init Window Manager
   _InitMenus                   ; Init Menu Manager
   Clr.l  -(SP)                 ; No restart Procedure
   _InitDialogs                 ; Init Dialog Manager
   _TEInit                      ; Init Text Edit
   _InitCursor                  ; Turn on arrow cursor
   Rts

;--------------------- SetupMenu ---------------

SetupMenu

; Apple Menu Set Up.

   ; FunctionGetMenu (menu ID:Integer): MenuHandle;
   Clr.l     -(SP)              ; Space for menu handle
   Move      #AppleMenu,-(SP)   ; Apple menu resource ID
   _GetRMenu                    ; Get menu handle
   Move.l    (SP),AppleHReg     ; Save for later comparison
   Move.l    (SP),-(SP)         ; Copy handle for AddResMenu

   Clr   -(SP)              ; Append to menu
   _InsertMenu                  ; Which is currently empty

   Move.l    #'DRVR',-(SP)      ; Load all drivers
   _AddResMenu                  ; And Add to Apple menu

; File Menu Set Up

   Clr.l     -(SP)              ; Space for menu handle
   Move      #FileMenu,-(SP)    ; File Menu Resource ID
   _GetRMenu                    ; Get File menu handle

   Clr   -(SP)              ; Append to list
   _InsertMenu                  ; After Apple menu

; Edit Menu Set Up

   Clr.l     -(SP)              ; Space for menu handle
   Move      #EditMenu,-(SP)    ; Edit menu resource ID
   _GetRMenu                    ; Get handle to menu
   Move.l    (SP),EditHReg      ; Save for later
                        ; Leave on stack for Insert

   Clr   -(SP)              ; Append to list
   _InsertMenu                  ; After File menu

; Font Menu Set Up

   Clr.l     -(SP)              ; Space for menu handle
   Move      #FontMenu,-(SP)    ; Font Menu Resource ID
   _GetRMenu                    ; Get File menu handle

   Clr   -(SP)          ; Append to list
   _InsertMenu              ; After Edit menu

   _DrawMenuBar             ; Display the menu bar
   Rts

;---------------- SetupWindow ---------------

SetupWindow

   ; FunctionGetNewWindow (windowID: Integer; wStorage: Ptr;
   ;                behind: WindowPtr) : WindowPtr;
   Clr.l     -(SP)              ; Space for window pointer
   Move      #ASample,-(SP)     ; Resource ID for window
   Pea       WindowStorage(A5)  ; Storage for window
   Move.l    #-1,-(SP)          ; Make it the top window
   _GetNewWindow                ; Draw the window
   Move.l    (SP),WindowPReg    ; Save for later

   _SetPort                     ; Make it the current port
   Rts

;------------- SetupTextEdit ----------------

SetupTextEdit

   ; Procedure  TENew (destRect,viewRect: Rect): TEHandle;
   Clr.l     -(SP)              ; Space for text handle
   Pea       DestRect           ; DestRect Rectangle
   Pea       ViewRect           ; ViewRect Rectangle
   _TENew                       ; New Text Record
   Move.l    (SP)+,TextHReg     ; Save text handle
   Rts

;------------- Event Handling Routines --------

HandleEvent

   Use the event number as an index into the Event table.
These 12 events are all the things that could spontaneously
happen while the program is in the main loop.

   Move      Modify,ModifyReg   ; More useful in a reg
   Move      What,D0            ; Get event number
   Add       D0,D0              ; *2 for table index
   Move      EventTable(D0),D0  ; Point to routine offset
   Jmp       EventTable(D0)     ; and jump to it

EventTable

   Dc.w      NextEvent-EventTable   ; Null Event (Not used)
   Dc.w      MouseDown-EventTable   ; Mouse Down
   Dc.w      NextEvent-EventTable   ; Mouse Up (Not used)
   Dc.w      KeyDown-EventTable     ; Key Down
   Dc.w      NextEvent-EventTable   ; Key Up (Not used)
   Dc.w      KeyDown-EventTable     ; Auto Key
   Dc.w      Update-EventTable      ; Update
   Dc.w      NextEvent-EventTable   ; Disk (Not used)
   Dc.w      Activate-EventTable    ; Activate
   Dc.w      NextEvent-EventTable   ; Abort (Not used)
   Dc.w      NextEvent-EventTable   ; Network (Not used)
   Dc.w      NextEvent-EventTable   ; I/O Driver (Not used)

;-------------------- Event Actions ----------------
Activate
```


© The Definitive MacTutor, Vol. 4

An activate event is posted by the system when a window needs to be activated or deactivated. The information that indicates which window needs to be updated was returned by the NextEvent call.

```
    Cmp.l  Message,WindowPReg      ; Was it our window?
    Bne NextEvent              ; No, get next event
    Btst   #activeFlag,ModifyReg   ; Activate?
    Beq Deactivate                 ; No, go do Deactivate
```

To activate our window, activate TextEdit, and disable Undo since we don't support it. Then set our window as the port since an accessory may have changed it. This activate event was generated by SelectWindow as a result of a click in the content region of our window. If the window had scroll bars, we would do ShowControl and HideControl here too.

```
    ; Procedure  TEActivate (hTE: TEHandle);
    Move.l     TextHReg,-(SP)      ; Move Text Handle To Stack
    _TEActivate                    ; Activate Text

    ; Procedure DisableItem (menu:MenuHandle; item:Integer);
    Move.l     EditHReg,-(SP)      ; Get handle to the menu
    Move       #UndoItem,-(SP)     ; Enable 1st item (undo)
    _DisableItem

SetOurPort                         ; used by InAppleMenu

    Move.l     WindowPReg,-(SP)    ; an accessory might have
    _SetPort                         ; changed it.

NextEvent
    Moveq #0,D0                      ; Say that it's not Quit
    Rts                              ; return to EventLoop

Deactivate

    Move.l     TextHReg,-(SP)      ; Get Text Handle
    _TeDeActivate                  ; Un Activate Text
    Move.l     EditHReg,-(SP)      ; Get handle to the menu
    Move       #UndoItem,-(SP)     ; Enable 1st item (undo)
    _EnableItem
    Bra    NextEvent           ; Go get next event

Update
```

The window needs to be redrawn. Erase the window and then call TextEdit to redraw it.

```
    ; Procedure  BeginUpdate (theWindow: WindowPtr);
    Move.l       WindowPReg,-(SP) ; Get pointer to window
    _BeginUpDate                  ; Begin the update

    ; EraseRect (rUpdate: Rect);
    Pea        ViewRect            ; Erase visible area
    _EraseRect

    ; TEUpdate (rUpdate: Rect; hTE: TEHandle);
    Pea        ViewRect            ; Get visible area
    Move.l     TextHReg,-(SP)      ; and handle to text
    _TEUpdate                      ; then update the window

    ; Procedure  EndUpdate (theWindow: WindowPtr);
    Move.l       WindowPReg,-(SP) ; Get pointer to window
    _EndUpdate                    ; and end the update
    Bra        NextEvent         ; Go get next event

KeyDown
```

A key was pressed. First check to see if it was a command key. If so, go do it. Otherwise pass the key to TextEdit.

```
    Btst       #CmdKey,ModifyReg   ; Is command key down?
    Bne    CommandDown         ; handle command key

    ; Procedure  TEKey (key: CHAR; hTE: TEHandle);
    Move       Message+2,-(SP)     ; Get character
    Move.l     TextHReg,-(SP)      ; and text record
    _TEKey                         ; Give char to TextEdit
    Bra    NextEvent           ; Go get next event

CommandDown
```

The command key was down. Call MenuKey to find out if it was the command key equivalent for a menu command, pass the menu and item numbers to Choices.

```
    ; FunctionMenuKey (ch:CHAR): LongInt;
    Clr.l      -(SP)               ; Space for Menu and Item
    Move       Message+2,-(SP)     ; Get character
    _MenuKey                       ; See if it's a command
    Move       (SP)+,MenuReg       ; Save Menu
    Move       (SP)+,MenuItemReg   ; and Menu Item
    Bra    Choices             ; Go dispatch command


;---------Mouse Down Events And Their Actions-----

MouseDown
```

If the mouse button was pressed, we must determine where the click occurred before we can do anything. Call FindWindow to determine where the click was; dispatch the event according to the result.

```
    ; FunctionFindWindow (thePt: Point;
    ;                     Var whichWindow: WindowPtr): Integer;
    Clr    -(SP)               ; Space for result
    Move.l     Point,-(SP)         ; Get mouse coordinates
    Pea        WWindow             ; Event Window
    _FindWindow                    ; Who's got the click?
    Move       (SP)+,D0            ; Get region number
    Add        D0,D0               ; *2 for index into table
    Move       WindowTable(D0),D0  ; Point to routine offset
    Jmp        WindowTable(D0)     ; Jump to routine

WindowTable

    Dc.w       NextEvent-WindowTable  ; In Desk (Not used)
    Dc.w       InMenu-WindowTable     ; In Menu Bar
    Dc.w       SystemEvent-WindowTable ; System Window
    Dc.w       Content-WindowTable    ; In Content
    Dc.w       Drag-WindowTable       ; In Drag
    Dc.w       NextEvent-WindowTable  ; In Grow (Not used)
    Dc.w       NextEvent-WindowTable  ; In Go Away (Not used)

SystemEvent
```

The mouse button was pressed in a system window. SystemClick calls the appropriate desk accessory to handle the event.

```
    ; Procedure  SystemClick (theEvent: EventRecord;
    ;                         theWindow: WindowPtr);
    Pea        EventRecord     ; Get event record
    Move.l     WWindow,-(SP)   ; and window pointer
    _SystemClick               ; Let the system do it
    Bra    NextEvent       ; Go get next event

Content
```

The click was in the content area of a window. If our window was in front, then call Quickdraw to get local coordinates, then pass the coordinates to TextEdit. We also determine whether the shift key was pressed so TextEdit can do shift-clicking. If our window wasn't in front, Move it to the

front, but don't process click.

```
   Clr.l       -(SP)           ; clear room for result
   _FrontWindow                ; get FrontWindow
   Move.l      (SP)+,D0        ; Is front window pointer
   Cmp.l       WindowPReg,D0   ; same as our pointer?
   Beq.s       @1              ; Yes, call TextEdit
```

We weren't active, select our window. This causes an activate event.

```
   ; Procedure  SelectWindow (theWindow: WindowPtr);
   Move.l      WWindow,-(SP)   ; Window Pointer To Stack
   _SelectWindow               ; Select Window
   Bra      NextEvent       ; and get next event

@1

; We were active, pass the click (with shift) to TextEdit.

   ; Procedure  GlobalToLocal (Var pt:Point);
   Pea      Point              ; Mouse Point
   _GlobalToLocal              ; Global To Local

   ; Procedure  TEClick (pt: Point; extend: Boolean; hTE:
TEHandle);
   Move.l   Point,-(SP)        ; Mouse Point (GTL)
   Btst     #shiftKey,ModifyReg ; Is shift key down?
   Sne      D0                 ; True if shift down
```

Note: We want the Boolean in the high byte, so use Move.b. The 68000 pushes an extra, unused byte on the stack for us.

```
   Move.b      D0,-(SP)
   Move.l      TextHReg,-(SP)  ; Identify Text
   _TEClick                    ; TEClick
   Bra      NextEvent       ; Go get next event

Drag

   Move.l   WWindow,-(SP)   ; Pass window pointer
   Move.l   Point,-(SP)     ; mouse coordinates
   Pea      WBounds         ; and boundaries
   _DragWindow              ; Drag Window
   Bra   NextEvent       ; Go get next event

InMenu

   Clr.l    -(SP)           ; Get Space For Menu Choice
   Move.l   Point,-(SP)     ; Mouse At Time Of Event
   _MenuSelect              ; Menu Select
   Move     (SP)+,MenuReg   ; Save Menu
   Move     (SP)+,MenuItemReg ; and Menu Item
```

On entry to Choices, the resource ID of the Menu is saved in the low word of a register, and the resource ID of the MenuItem in another. The routine MenuKey, used when a command key is pressed, returns the sameinfo.

```
Choices                     ; Called by command key too

   Cmp #AppleMenu,MenuReg  ; Is It In Apple Menu?
   Beq InAppleMenu         ; Go do Apple Menu
   Cmp #FileMenu,MenuReg   ; Is It In File Menu?
   Beq InFileMenu          ; Go do File Menu
   Cmp #EditMenu,MenuReg   ; Is It In Edit Menu?
   Beq InEditMenu          ; Go do Edit Menu
   Cmp #FontMenu,MenuReg   ; Is It In Font Menu?
   Beq InFontMenu          ; Go do Font Menu

ChoiceReturn

   Bsr         UnHiliteMenu       ; Unhighlight the menu
bar
   Bra         NextEvent          ; Go get next event

InFileMenu

   Cmp      #QuitItem,MenuItemReg  ; Is It Quit?
   Bne   ChoiceReturn          ; No, Go get next event
   Bsr      UnHiliteMenu       ; Unhighlight the menu bar
   Move     #-1,D0             ; say it was Quit
   Rts

InEditMenu

; First, call SystemEdit. If a desk accessory is active that
uses the Edit menu (such as the Notepad) this lets it use our
menu. Decide whether it was cut, copy, paste, or clear.
Ignore Undo since we didn't implement it.

   Bsr   SystemEdit          ; Desk accessory active?
   Bne ChoiceReturn        ; Yes, SystemEdit handled it
   Cmp   #CutItem,MenuItemReg; Is It Cut?
   Beq      Cut               ; Yes, go handle it
   Cmp      #CopyItem,MenuItemReg  ; Is it Copy?
   Beq      Copy              ; Yes, go handle it
   Cmp      #PasteItem,MenuItemReg ; Is it Paste?
   Beq      Paste             ; Yes, go handle it
   Cmp      #ClearItem,MenuItemReg ; Is it Clear?
   Beq      ClearIt           ; Yes, go handle it
   Bra   ChoiceReturn        ; Go get next event

InFontMenu

   Cmp      #DialogItem,MenuItemReg; Is It Dialog?
   Bne   ChoiceReturn           ; No, Go get next event

   ; Procedure  TEDeActivate (hTE: TEHandle)
   Move.l      TextHReg,-(SP)      ; Identify Text
   _TEDeActivate                   ; Deactivate Text

; Obtain the current Font number, size and attributes to setup
the dialog box.
   Movea.l  TextHReg,A0      ; Handle to the TE record
   Movea.l  (A0),A0          ; Pointer to the TE record
   Move.w   teFont(A0),D0    ;- Font Number
   Cmp.w    #1,D0            ;   Default number?
   Bne.s    @0               ;   No, -> @0
   Move.w   ApFontID,D0      ;   Yes, load default number
@0 Move.w   D0,DtxFont(A5)
   Clr.w    D0               ;- Font Face
   Move.b   teFace(A0),D0    ;   Place the attribute bits in
the
   Move.w   D0,DtxFace(A5)   ;   low byte of DtxFace(A5).
   Move.w   teSize(A0),D0    ;- Font Size
   Bne.s    @1               ;   Default size, No -> @1
   Move.b   FMDefaultSize,D0 ;  Yes, load default size
@1 Move.w   D0,DtxSize(A5)

   Clr.w    -(SP)
   Pea      DtxFont(A5)  ; Load the font's number address
   Pea      DtxFace(A5)  ; Load the font's face address
   Pea      DtxSize(A5)  ; Load the font's size address
   Jsr      FontDialog
```

Test to see if the okButton was pressed (ie install the new Font variables). If the cancelButton was pressed leave the Font variables as they are.

```
   Move.b   (SP)+,D0
   Beq.s    @2
```

Update the Font number, size, and face fields in the textedit record. The teAscent and teLineHite fields also require up dating. Reset the port to the textedit window, since the current GraphPort has been changed by the FontDia-

© The Definitive MacTutor, Vol. 4

```
log routine.

   Move.l       WindowPReg,-(SP)
   _SetPort
   Move.w    DtxFont(A5),-(SP)
   _TextFont
   Move.w    DtxFace(A5),-(SP)
   _TextFace
   Move.w    DtxSize(A5),-(SP)
   _TextSize

   ; Procedure GetFontInfo (Var Info: FontInfo)
   Pea       Info(A5)
   _GetFontInfo

   Movea.l   TextHReg,A1      ; DeReference the Handle
   Movea.l   (A1),A1
   Move.w    DtxFont(A5),teFont(A1) ; teFont
   Move.b    DtxFace+1(A5),teFace(A1)   ; teFace
   Move.w    DtxSize(A5),teSize(A1)     ; teSize
   Move.w    Info+ascent(A5),teAscent(A1)  ; teAscent
   Move.w    Info+ascent(A5),D0         ; teLineHite
   Add.w     Info+descent(A5),D0
   Add.w     Info+leading(A5),D0
   Move.w    D0,teLineHite(A1)

@2 Bra    GoSetOurPort

InAppleMenu

   Cmp    #AboutItem,MenuItemReg ; Is It About?
   Beq About                     ; If So Goto About...

   ; Procedure  GetItem (menu: MenuHandle; item: Integer;
   ;                        Var itemString: Str255);
   Move.l    AppleHReg,-(SP)     ; Look in Apple Menu
   Move      MenuItemReg,-(SP)   ; What Item Number?
   Pea       DeskName            ; Get Item Name
   _GetItem                      ; Get Item

   ; FunctionOpenDeskAcc (theAcc: Str255) : Integer;
   Clr    -(SP)                  ; Space For Opening Result
   Pea       DeskName            ; Open Desk Acc
   _OpenDeskAcc                  ; Open It
   Move      (SP)+,D0            ; Pop result

GoSetOurPort

   Bsr       SetOurPort             ; Set port to us
   Bra          ChoiceReturn        ; Unhilite menu and
return

;---------- Text Editing Routines ---------

Cut               ; CUT
   Move.l    TextHReg,-(SP)      ; Identify Text
   _TECut                        ; Cut it and copy it
   Bra    ChoiceReturn        ; Go get next event
Copy              ; COPY
   Move.l    TextHReg,-(SP)      ; Identify Text
   _TECopy                       ; Copy text to clipboard
   Bra    ChoiceReturn        ; Go get next event
Paste             ; PASTE
   Move.l    TextHReg,-(SP)      ; Identify Text
   _TEPaste                      ; Paste
   Bra    ChoiceReturn        ; Go get next event
ClearIt                ;CLEAR
   Move.l    TextHReg,-(SP)      ; Point to text
   _TEDelete                     ; Clear without copying
   Bra    ChoiceReturn        ; Go get next event

   SystemEdit does undo, cut, copy, paste, and clear for desk
accessories. It returns False (Beq) if the active window
doesn't belong to a desk accessory.

SystemEdit

   ; FunctionSystemEdit (editCmd:Integer): Boolean;
   Clr    -(SP)                  ; Space for result
   Move      MenuItemReg,-(SP)   ; Get item in Edit menu
   Subq      #1,(SP)             ; SystemEdit is off by 1
   _SysEdit                      ; Do It
   Move.b (SP)+,D0            ; Pop result
   Rts                           ; Beq if NOT handled

UnhiliteMenu

   ; Procedure  HiLiteMenu (menuID: Integer);
   Clr       -(SP)               ; All Menus
   _HiLiteMenu                   ; UnHilite Them All
   Rts


;--------------------Misc Routines------------

About



   ; FunctionGetNewDialog (dialogID: Integer; dStorage: Ptr;
   ;                           behind: WindowPtr) : Dia-
logPtr
   Clr.l     -(SP)               ; Space For dialog pointer
   Move      #AboutDialog,-(SP)  ; Identify dialog rsrc #
   Pea       DStorage            ; Storage area
   Move.l    #-1,-(SP)           ; Dialog goes on top
   _GetNewDialog                 ; Display dialog box
   Move.l    (SP),-(SP)          ; Copy handle for Close
                                 ; Handle on stack
   _SetPort                      ; Make dialog box the port
   Move.l       TextHReg,-(SP)   ; Identify Text
   _TEDeActivate                 ; Deactivate Text

WaitOK

   ; Procedure  ModalDialog (filterProc: ProcPtr;
   ;              Var itemHit: Integer);
   Clr.l     -(SP)               ; Clear space For handle
   Pea       ItemHit             ; Storage for item hit
   _ModalDialog                  ; Wait for a response

   Move      ItemHit,D0          ; Look to see what was hit
   Cmp       #ButtonItem,D0      ; was it OK?
   Bne    WaitOK              ; No, wait for OK

   _CloseDialog                  ; Handle already on stack
    Bra      GoSetOurPort        ; Set port to us and return

; ---------- Data Starts Here ------------

EventRecord                            ; NextEvent's Record
 What:       Dc       0              ; Event number
 Message:       Dc.l      0              ; Additional informa-
tion
 When:          Dc.l      0              ; Time event was
posted
 Point:         Dc.l      0              ; Mouse coordinates
 Modify:        Dc        0              ; State of keys and
button
 WWindow:       Dc.l      0              ; Find Window's Result

DStorage       Dcb.w     DWindLen,0     ; Storage For Dialog
DeskName       Dcb.w     16,0           ; Desk Accessory's
Name
WBounds        Dc        28,4,308,508 ; Drag Window's Bounds
ViewRect       Dc        5,4,245,405  ; Text Record's View
```

```
Rect
DestRect        Dc      5,4,245,405  ; Text Record's Dest
Rect
ItemHit         Dc      0            ; Item clicked in
dialog

;-------------- Nonrelocatable Storage ---------

   Variables declared using Ds are placed in a global space
relative to A5.  When these variables are referenced, A5 must
be explicitly mentioned.

DtxFont         Ds.w    1
DtxFace         Ds.w    1
DtxSize         Ds.w    1
Info            Ds.w    4            ; FontInfo Record
WindowStorage   Ds.w    WindowSize   ; Storage for Window

End

;-------------- FontDialog --------------------

; Written by Ray.A.Cameron
; Version 1
;Wed May 6, 1988 21:00:51


XDef        FontDialog      ; Routine's name
XRef        FontMap         ; Variable's name

;---------- Includes -------------

Include     Traps.D         ; Use System and ToolBox traps
Include     ToolEqu.D       ; Use ToolBox Equates
Include     QuickEqu.D      ; Use QuickDraw Equates
Include     SysEqu.D        ; Use System Equates
Include     PackMacs.Txt    ; Use Package Equates
Include     FontMacros.Txt  ; Use Macro file


; ---------- Equates ----------
FontDPtr    EquA4

FDialog     Equ260   ; Font dialog is DLOG resource #260
ADialog     Equ260   ; Alert dialog is ALRT resource #260
True        Equ 1    ; Boolean True
False       Equ0     ; Boolean False
enterCode   Equ3     ; ASCII Enter
bsCode      Equ8     ; ASCII Back Space
crCode      Equ 13   ; ASCII Carriage Return
param0      Equ0     ; Offset to string 0 in DAStrings
param1      Equ4     ; Offset to string 1 in DAStrings
param2      Equ8     ; Offset to string 2 in DAStrings
OkActive    Equ0     ; OkActive bit within Dialog Flags
SUpdate     Equ 1    ; Sample update bit within Dialog Flags
SRect       Equ2     ; Sample rect bit within Dialog Flags
SKey        Equ3     ; Sample key bit within Dialog Flags

; --------- Font Dialog Data Record -------

DStorage    Equ$0    ; Storage for the Dialog Record
TempFont    Equ$AA   ; Currently selected font number
TempFace    Equ$AC   ; Currently selected font style
TempSize    Equ$AE   ; Currently selected font size
DFlags      Equ$B0   ; Dialog Flags
NameListH   Equ$B2   ; Font Name Selection List's Handle
NameCell    Equ$B6   ; Font Name selected cell
SizeListH   Equ$BA   ; Font Size Selection List's Handle
SizeCell    Equ$BE   ; Font Size selected cell
TempCell    Equ$C2   ; Temporary Cell variable
Item        Equ$C6   ; Item's handle or procedure
Itemtype    Equ$CA   ; Item's type
ItemBox     Equ$CC   ; Item's enclosing rect
ItemHit     Equ$D4   ; Item clicked in dialog
loc_pt      Equ$D6   ; mouse location
InitBounds  Equ$DA   ; Init size of Lists
TextCursor  Equ$E2   ; Handle to the IBeam cursor
thename     Equ$E6   ; Text string
pnState     Equ$1E6  ; Original state of the pen
String1     Equ$1F8  ; Alert text string1
String2     Equ$1FC  ; Alert text string2
String3     Equ$200  ; Alert text string3
Sample      Equ$204  ; Sample Text Edit Handle
SampleRect  Equ$208  ; Sample Text Edit Rect
SampleTicks Equ$210    ; Ticks when Sample required
updated
info        Equ$214    ; GetFontInfo record
search      Equ$21C    ; String to search Font Name List

GlobalslengthEqu$23C

; This routine receives the following inputs:
;  teFont: The current font's number
;  teFace: The current type face's attributes in low byte
;     Bit 0 - Bold        Bit 3 - Outline     Bit 6 - Extend
;     Bit 1 - Italic      Bit 4 - Shadow
;     Bit 2 - Underline   Bit 5 - Condense
;  teSize: The current font's size

   A ModalDialog box  is displayed to allow the user to choose
the font attributes that they desire.  If the Ok button, enter
key or carriage return is pressed the selected values will
replace the original values otherwise the values remain the
same.


FontDialog

   Link   A6,#0
   ; Boolean result    20(A6) ; Ok or Cancel
   ; DtxFont Address   16(A6) ; Address of current Font number
   ; DtxFace Address   12(A6) ; Address of current Font style
   ; DtxSize Address   8(A6)  ; Address of current Font size
   ; Return Address    4(A6)
   ; Old A6            (A6)

   Movem.l   D0-D7/A0-A4,-(SP)   ; Save register values.
   Move.w    #False,20(A6)       ; Set up to return False.

   ; Create a non-relocatable block for Font Data Record.
   Move.l       #Globalslength,D0
   _NewPtr      ,CLEAR
   Move.l       A0,FontDRec(A5)     ; Store the pointer.
   Movea.l      A0,FontDPtr

   GetPenState  pnState(A4)  ; Save the current pen settings.
   ; Load the present values of the font number, face and size
   ; into temporary locations.
   Movea.l   16(A6),A0           ; Font number.
   Move.w    (A0),TempFont(A4)
   Movea.l   12(A6),A0           ; Font face.
   Move.w    (A0),TempFace(A4)
   Movea.l   8(A6),A0            ; Font size.
   Move.w    (A0),TempSize(A4)

   ; Obtain the iBeamCursor's resource and lock it down.
   ; Function GetCursor (cursorID: Integer): CursHandle
   Subq.l    #4,SP
   Move.w    #iBeamCursor,-(SP)
   _GetCursor
   Move.l    (SP)+,TextCursor(A4)
   HLock     TextCursor(A4)

   Subq.l    #4,SP           ; Space for dialog pointer.
   Move.w #FDialog,-(SP)   ; Identify dialog rsrc #.
   Move.l    FontDPtr,-(SP)  ; Storage area.
```


© The Definitive MacTutor, Vol. 4

```
   Move.l    #-1,-(SP)        ; Dialog goes on top.
   _GetNewDialog
   _SetPort

   Move.l    OneOne,InitBounds+bottom(A4)
   Bset.b    #0kActive,DFlags(A4) ; selections are valid.

   ; Lock FontMap while its information is being accessed.
   HLock     FontMap(A5)
   DeRefHndleFontMap(A5),A3

   Set up the Style check boxes so that they show the current
font's values.

   Move.w    TempFace(A4),D0      ; current face (style) byte.
   Move.w    #4,-(SP)             ; Item No 4.
   Move.w    #5,-(SP)             ; 5 Items (check boxes).
   Move.w    D0,-(SP)             ; Bits to set.
   Bsr       SControl

   Set up the Spacing radio buttons so that they show the
current fonts values.

   Move.w    TempFace(A4),D0  ; current face (style) byte.
   Andi.w    #%01100000,D0    ; Examine Condense & Extend.
   Bne.s     @0               ; If neither are set, then
   Move.w    #%00010000,D0    ; add bit to represent "Normal".
@0 Lsr.W        #4,D0

   Move.w    #11,-(SP)        ; Item No 11.
   Move.w    #3,-(SP)         ; 3 Items (radio buttons).
   Move.w    D0,-(SP)         ; Bits to set.
   Bsr       SControl

   ; Set up the Font Name Selection Window

   GetDItem  FontDPtr,#17,Itemtype(A4),Item(A4),ItemBox(A4)

   Subq.l    #4,SP                ; Space for the handle.
   Pea       ItemBox(A4)
   Pea       InitBounds(A4)       ; Create one cell.
   Clr.l     -(SP)                ; Cell size for text.
   Move.w    #0,-(SP)             ; Text definition procedure.
   Move.l    FontDPtr,-(SP)
   Move.b    #False,-(SP)         ; drawIt
   Move.b    #False,-(SP)         ; hasGrow
   Move.b    #False,-(SP)         ; scrollHoriz
   Move.b    #True,-(SP)          ; scrollVert
   _LNew

   Move.l    (SP),NameListH(A4)   ; Store the List's handle.
   DeRefHndle(SP)+,A0             ; Set the selection flags.
   Move.b    #%10000010,selFlags(A0)  ; 10nlyOne, 1NoNilHil-
ite
   Move.l    #$FFFF0000,NameCell(A4)  ; (0,-1), no cell

   Jsr       LdNames              ; Load the Font Names.

; Set up the Font Size Selection Window

   GetDItem  FontDPtr,#19,Itemtype(A4),Item(A4),ItemBox(A4)

   ; Function LNew (rView, dataBounds: Rect; cSize: Point;
   ;  theProc: Integer; theWindow: WindowPtr; drawIt, hasGrow,
   ;  scrollHoriz, scrollVert: Boolean): ListHandle

   Subq.l    #4,SP                ; Space for the handle.
   Pea       ItemBox(A4)
   Pea       InitBounds(A4)       ; Create one cell.
   Clr.l     -(SP)                ; Cell size for text.
   Move.w    #0,-(SP)             ; Text definition procedure.
   Move.l    FontDPtr,-(SP)
   Move.b    #False,-(SP)         ; drawIt
   Move.b    #False,-(SP)         ; hasGrow
   Move.b    #False,-(SP)         ; scrollHoriz
   Move.b    #True,-(SP)          ; scrollVert
   _LNew

   Move.l    (SP),SizeListH(A4)   ; Store the List's handle.
   DeRefHndle(SP)+,A0             ; Set the selection flags.
   Move.b    #%10000010,selFlags(A0)  ; 10nlyOne, 1NoNilHil-
ite
   Move.l    #$FFFF0000,SizeCell(A4)

   Jsr       LdSizes              ; Load the current Font's Sizes.

; Set up the Font Size Text Edit item

   Jsr          SetEditSize

; Set up the Sample Display Text Edit Record

   GetDItem  FontDPtr,#21,Itemtype(A4),Item(A4),SampleRect(A4)
   ; Function TENew (destRect, viewRect: Rect): TEHandle
   Subq.l    #4,SP
   Pea    SampleRect(A4)   ; Create a new text edit record.
   Pea    SampleRect(A4)
   _TENew
   Move.l    (SP)+,Sample(A4)     ; Store theTEdit handle.
   HLock     Sample(A4)
   DeRefHndleSample(A4),A0        ; Setup the variables.
   Move.w    #teJustCenter,teJust(A0)
   Move.b    MinusOne,teCROnly(A0)

   GetResource  'STR ',#264  ; Obtain the sample text.
   HLock        (SP)         ; Leave the handle on the stack.
   DeRefHndle(SP),A0         ; Address of text in A0.
   Clr.l        D0           ; Length of text in D0.
   Move.b       (A0)+,D0

   Move.l       A0,-(SP)     ; Insert the sample string into
   Move.l       D0,-(SP)     ; the record.
   Move.l       Sample(A4),-(SP)
   _TEInsert
   HUnlock      (SP)+        ; Unlock the string resource.

   Jsr          SampleWindow ; Update the TEdit variables.
   HUnlock      Sample(A4)

; Set up the Alert Dialog Variables
   Move.w    #-1,ACount       ; Reset the Alert stage to 1.
   GetResource  'STR ',#261  ; Obtain copies of the Alert
strings
   Move.l (SP)+,String1(A4)  ; handles and store for easy
   GetResource  'STR ',#262  ; access.
   Move.l       (SP)+,String2(A4)
   GetResource  'STR ',#263
   Move.l       (SP)+,String3(A4)

; Display the Dialog Window
   ShowWindowFontDPtr      ; This posts an Activate event.


WaitOk
   Pea       DUserFilter
   Pea       ItemHit(A4)
   _ModalDialog

   Move.w    ItemHit(A4),D0  ; Get the itemNo that was Hit.
   Subq.w    #1,D0           ; Allow for no item zero.
   Lsl.w     #1,D0           ; *2 for table index.
   Move.w    ItemTable(D0),D0 ; Point to routine offset
   Jmp       ItemTable(D0)   ; and jump to it.

ItemTable
   Dc.w      DOk-ItemTable       ; Ok         Button
```

```
   Dc.w      DCancel-ItemTable   ; Cancel      Button
   Dc.w      WaitOk-ItemTable    ; Character… Text (Disabled)
   Dc.w      SStyle-ItemTable    ; Bold    Check Box
   Dc.w      SStyle-ItemTable    ; Italic      Check Box
   Dc.w      SStyle-ItemTable    ; Underline   Check Box
   Dc.w      SStyle-ItemTable    ; Outline Check Box
   Dc.w      SStyle-ItemTable    ; Shadow      Check Box
   Dc.w      Style-ItemTable     ; Style       Text (Enabled)
   Dc.w      WaitOk-ItemTable    ; ThinBox User Item
   Dc.w      SSpacing-ItemTable  ; Normal      Radio Button
   Dc.w      SSpacing-ItemTable  ; Condense    Radio Button
   Dc.w      SSpacing-ItemTable  ; Extend      Radio Button
   Dc.w      Spacing-ItemTable   ; Spacing Text (Enabled)
   Dc.w      WaitOk-ItemTable    ; ThinBox User Item
   Dc.w      WaitOk-ItemTable    ; Font N...   Text (Disabled)
   Dc.w      FontName-ItemTable  ; NameSel Wind  User Item
   Dc.w      WaitOk-ItemTable    ; Font S...   Text (Disabled)
   Dc.w      FontSize-ItemTable  ; SizeSel Wind  User Item
   Dc.w      GetEditSize-ItemTable  ; Type size  Text Edit
   Dc.w      FontSample-ItemTable; Sample Font   User Item

DOk
   ; If the current Dialog selection isn't valid, set up the
appropriate param text strings and invoke the Caution Alert.
   Btst.b    #OkActive,DFlags(A4); Is the selection valid?
   Bne       @3                  ; Yes -> @3
   Clr.w     D0                  ; Set up the three param
   Movea.l   #DAStrings,A0       ; text strings.
   Clr.l     param0(A0)
   Clr.l     param1(A0)
   Clr.l     param2(A0)
   Cmpi.w    #-1,TempFont(A4) ; Is TempFont a valid selection?
   Bne.s     @0              ; Yes -> @0
   Bset.b    #0,D0           ; No -> include related text.
   Move.l    String1(A4),param0(A0)
@0 Cmpi.w    #0,TempSize(A4)  ; Is TempSize a valid selection?
   Bne.s     @1              ; Yes -> @1
   Bset.b    #1,D0           ; No -> include related text.
   Move.l    String2(A4),param2(A0)
@1 Cmpi.w    #3,D0  ; If both TempFont and TempSize
   Bne.s     @2     ; are invalid include the conjunctive
   Move.l    String3(A4),param1(A0) ; string (String3).
@2 ; Function CautionAlert (alertID: Integer; filterProc:
ProcPtr): Integer
   Subq.l    #2,SP
   Move.w    #ADialog,-(SP)
   Clr.l     -(SP)
   _CautionAlert
   Addq.l    #2,SP
   Bra       WaitOk    ; User to alter their selection.

@3 ;
   If the current Dialog selection is valid, replace the
original font number, style and size values with the values
chosen by the user.

   Movea.l   16(A6),A0           ; Font number.
   Move.w    TempFont(A4),(A0)
   Movea.l   12(A6),A0           ; Font style.
   Move.w    TempFace(A4),(A0)
   Movea.l   8(A6),A0            ; Font size.
   Move.w    TempSize(A4),(A0)
   Move.b    #True,20(A6)        ; Return True.

DCancel
   HideWindowFontDPtr
   LMDispose    NameListH(A4)    ; Font Name List.
   LMDispose    SizeListH(A4)    ; Font Size List.
   TEDispose    Sample(A4)       ; Sample Text Edit Record.
   CloseDialog  FontDRec(A5)
   HUnLock      FontMap(A5)      ; FontMap .
   HUnLock      TextCursor(A4)   ; iBeamCursor resource.
   SetPenState  pnState(A4)      ; Restore the pen settings.
   DisposePtr   FontDPtr         ; Font Dialog Data Record.

   Movem.l   (SP)+,D0-D7/A0-A4   ; Restore registers.
   Unlk      A6
   Movea.l   (SP)+,A0            ; Pop args & return.
   Add.l     #12,SP
   Jmp       (A0)

Style
   Move.w    TempFace(A4),D2  ; Store for comparison later.
   Movea.l      12(A6),A0     ; Obtain original style values
   Move.w    (A0),D0          ; and retain lower five bits.
   Andi.w    #%00011111,D0
   Andi.w    #%11100000,TempFace(A4); Clear the lower 5 bits.
   Or.w         D0,TempFace(A4)  ; Combine the two together.
   Bra.s        SetStyle
SStyle
   Move.w    TempFace(A4),D2  ; Store for comparison later.
   Move.w    ItemHit(A4),D0   ; Get the itemNo that was hit.
   Subq.w    #4,D0
   Bchg.b    D0,TempFace+1(A4)   ; Toggle the selected item.
SetStyle
   Cmp.w     TempFace(A4),D2     ; Has the face altered?
   Beq.s     @1                  ; No -> @1
   Move.w    TempFace(A4),D0
   ; Routine SControl (StItem, NofItems, ValueBits : Integer)
   Move.w    #4,-(SP)            ; Item No 4.
   Move.w    #5,-(SP)            ; 5 Items (check boxes).
   Move.w    D0,-(SP)            ; Bits to set.
   Bsr    SControl
   Btst.b #OkActive,DFlags(A4); Are Dialog selections valid.
   Beq.s     @1          ; No, don't update the sample.
   Bclr.b    #SRect,DFlags(A4)   ; Erase Sample if it hasn't
   Beq.s     @0                  ; been already.
   EraseRect SampleRect(A4)
@0 Bset.b    #SUpdate,DFlags(A4) ; Window requires updating.
   Move.l    Ticks,SampleTicks(A4)
@1 Jmp       WaitOk

Spacing
   Move.w    TempFace(A4),D2  ; Store for comparison later.
   Movea.l   12(A6),A0     ; Obtain the original style values.
   Move.w    (A0),D0
   Bra.s     SetSpacing
SSpacing
   Move.w    TempFace(A4),D2  ; Store for comparison later.
   Move.w    ItemHit(A4),D1   ; Get the item No that was hit.
   Sub.w     #11,D1           ; Produce in D0 a byte that
   Move.w    #%00010000,D0    ; represents which bit to set.
   Lsl.b     D1,D0
SetSpacing
   Andi.w    #%11100000,D0    ; Clear the lower 5 bits.
   Andi.w    #%00011111,TempFace(A4); Clear the top 3 bits.
   Or.w      D0,TempFace(A4)  ; Combine the two together.
   Cmp.w     TempFace(A4),D2  ; Has the face altered?
   Beq.s     @2               ; No -> @2
   Move.w    TempFace(A4),D0  ; Produce in the first 3 bits
   Andi.w    #%01100000,D0    ; of D0 the settings of the
   Bne.s     @0               ; 3 radio buttons.
   Move.w    #%00010000,D0
@0 Lsr.b     #4,D0
   ; Routine SControl (StItem, NofItems, ValueBits : Integer)
   Move.w    #11,-(SP)           ; Item No 11.
   Move.w    #3,-(SP)            ; 3 Items (radio buttons).
   Move.w    D0,-(SP)            ; Bits to set.
   Bsr    SControl
   Btst.b #OkActive,DFlags(A4); Are Dialog selections valid.
   Beq.s     @2          ; No, don't update the sample.
   Bclr.b    #SRect,DFlags(A4); Erase Rect if it hasn't
   Beq.s     @1                  ; been already.
   EraseRect SampleRect(A4)
@1 Bset.b    #SUpdate,DFlags(A4) ; Window requires updating.
   Move.l    Ticks,SampleTicks(A4)
```

```
@2 Jmp      WaitOk


SControl
   ; Routine SControl (StItem, NofItems, ValueBits : Integer)

   Link           A6,#Ø
   ; StItem       12(A6)    Start Item number
   ; NofItems     10(A6)    Number of Items
   ; ValueBits    8(A6)     Values of the items
   ; Return address 4(A6)
   ; Old A6       (A6)

SetControl
   GetDItem
FontDPtr,12(A6),Itemtype(A4),Item(A4),ItemBox(A4)
   Clr.w    DØ           ; Set DØ to 1 if the item
   Lsr.W    8(A6)        ; is to be set or Ø if its not.
   Bcc.s    @1
   Addq.w   #1,DØ
@1 SetCtlValue  Item(A4),DØ
   Addq.w   #1,12(A6)    ; Increment the item No counter.
   Sub.w    #1,10(A6)    ; Any more items to process?
   Bne.s    SetControl   ; Yes -> SetControl.

   Unlk     A6
   Movea.l  (SP)+,AØ     ; Pop args & return.
   Add.l    #6,SP
   Jmp      (AØ)


FontName

   The user has made a new selection from the Font Name
Selection Window, either a new font name or none. TempFont
will change.

   Bclr.b   #SRect,DFlags(A4)   ; Erase Rect if it hasn't
   Beq.s    @Ø                  ; been already.
   EraseRect SampleRect(A4)
@Ø Move.w   NameCell+v(A4),DØ   ; Update the font number in
   Bmi.s    @1                  ; TempFont with the number
   Mulu     #6,DØ               ; of the font selected .
   Move.w   2(A3,DØ.W),TempFont(A4)
   Cmpi.w   #Ø,TempSize(A4)
   Beq.s    @2
   Bset.b   #OkActive,DFlags(A4); selections are valid.
   Bset.b   #SUpdate,DFlags(A4) ; Window requires updating.
   Move.l   Ticks,SampleTicks(A4)
   Bra.s    @3
@1 Move.w   #-1,TempFont(A4)    ; No Font Name is selected.
@2 Bclr.b   #OkActive,DFlags(A4); selections aren't valid.
@3 Jsr      LdSizes      ; Update Font Size Selection
Window.
   GetDItem  FontDPtr,#19,Itemtype(A4),Item(A4),ItemBox(A4)
   EraseRect ItemBox(A4)
   LMUpdate  VisRgn(A4),SizeListH(A4)
   Jmp       WaitOk


FontSize

   The user has made a new selection from the Font Size
Selection Window, either a new font name or none. TempSize
might not change.

   Move.w   TempSize(A4),D2     ; Store for comparison later.
   Move.w   NameCell+v(A4),DØ
   Bmi.s    @Ø                  ; No Font Name selected.
   Mulu     #6,DØ
   Move.w   2+4(A3,DØ.W),DØ  ; Offset to font's size list.
   Lea      (A3,DØ.W),AØ     ; Address of the size list.
   Move.w   SizeCell+v(A4),DØ
   Bmi.s    @Ø                  ; No Font Size selected.
   Lsl.w    #1,DØ
   Move.w   2(AØ,DØ.W),TempSize(A4); Store new font size.
   Cmp.w    TempSize(A4),D2     ; Has the size altered?
   Beq.s    @Ø                  ; No -> @Ø
   Jsr      SetEditSize         ; Update the textedit window.
@Ø Jmp      WaitOk


GetEditSize

   Convert the text in the font size text edit window into a
number to update TempSize. TempSize might not have changed, if
the mouse is clicked within  the font size text edit window
this routine is called.

   GetDItem  FontDPtr,#2Ø,Itemtype(A4),Item(A4),ItemBox(A4)
   GetIText  Item(A4),thename(A4)
   Lea       thename(A4),AØ
   _StringToNum
   Cmpi.w    #4,DØ
   Blt       @2                  ; if <4
   Cmpi.w    #127,DØ
   Bgt       @2                  ; if >127
   Cmp.w     TempSize(A4),DØ     ; Has the size altered?
   Beq.s     @1                  ; No -> @1
   Move.w    DØ,TempSize(A4)     ; Yes -> update TempSize.
   Cmp.w     #-1,TempFont(A4)    ; Is there a valid font name?
   Beq.s     @3                  ; No -> @3
   Bset.b    #OkActive,DFlags(A4); selections are valid.
   Bclr.b    #SRect,DFlags(A4)   ; Erase Rect if it hasn't
   Beq.s     @Ø                  ; been already.
   EraseRect SampleRect(A4)
@Ø Bset.b    #SUpdate,DFlags(A4) ; Window requires updating.
   Move.l    Ticks,SampleTicks(A4)
@1 Jmp       WaitOk

@2 Clr.w     TempSize(A4)    ; Indicate its outside limits.
@3 Bclr.b    #OkActive,DFlags(A4); selections aren't valid.
   Bclr.b    #SRect,DFlags(A4)   ; Erase Rect if it hasn't
   Beq.s     @4                  ; been already.
   EraseRect SampleRect(A4)
@4 Jmp       WaitOk


FontSample
   Jsr       SampleWindow        ; update the attributes
   TEUpdate  SampleRect(A4),Sample(A4)
   Bclr.b    #SUpdate,DFlags(A4) ; Window has been updated.
   Bset.b    #SRect,DFlags(A4)   ; Sample Rect isn't blank.
   Jmp       WaitOk


SetEditSize

   Setup the font size text edit item. To access this routine
there must be a valid font name.

   GetDItem     FontDPtr,#2Ø,Itemtype(A4),Item(A4),ItemBox(A4)
   Clr.l     D7
   Move.w    TempSize(A4),D7
   Cmpi.w    #4,D7
   Blt       @1                  ; if <4
   Cmpi.w    #127,D7
   Bgt       @1                  ; if >127
   Bset.b    #OkActive,DFlags(A4); selections are valid.
   Bclr.b    #SRect,DFlags(A4)   ; Erase Sample if it hasn't
   Beq.s     @Ø               ; been already.
   EraseRect SampleRect(A4)
@Ø Bset.b    #SUpdate,DFlags(A4) ; Window requires updating.
   Move.l    Ticks,SampleTicks(A4)
   Lea       thename(A4),AØ      ; Load thename pointer.
   Move.l    D7,DØ
```

```
   _NumToString
   Move.l    Item(A4),-(SP)  ; Set the item text to the new
   Move.l    A0,-(SP)        ; number string.
   _SetIText

   Move.l    FontDPtr,-(SP)
   Move.w    #20,-(SP) ; Select complete text in the item.
   Move.w    #0,-(SP)
   Move.w    #255,-(SP)
   _SelIText
   Rts

@1 Clr.w  TempSize(A4)       ; Indicate it's outside limits.
   Bclr.b    #0kActive,DFlags(A4); selections aren't valid.
   Bclr.b    #SRect,DFlags(A4)   ; Erase Sample if it hasn't
   Beq.s  @2                     ; been already.
   EraseRect SampleRect(A4)
@2 Rts


LdNames
   ; Load the NameList  with the font names stored in FontMap.
   ; Function LAddRow (count, rowNum: Integer; lHandle:
ListHandle): Integer
   Subq.l    #2,SP
   Move.w    (A3),-(SP)   ; The number of font names - 1.
   Move.w    #0,-(SP)
   Move.l    NameListH(A4),-(SP)
   _LAddRow
   Addq.l    #2,SP

   Clr.w     D7        ; Loop counter (the # of fonts -1).
@0 Move.w    D7,D6
   Mulu      #6,D6     ; Point to the next font info.
   Move.w    2(A3,D6.W),D0    ; Obtain the font's number.
   Cmp.w     TempFont(A4),D0  ; Is this font to be selected?
   Bne.s     @1               ; No -> @1.
   Move.w    D7,NameCell+v(A4)   ; cell # into NameCell+v.
@1 Move.w    2+2(A3,D6.W),D6  ; Obtain the font's name offset.

   ; Procedure LSetCell (dataPtr: Ptr; dataLen: Integer;
theCell: Cell; lHandle: ListHandle)
   Pea       1(A3,D6.W)   ; Address of the name (string).
   Clr.w     D0
   Move.b    (A3,D6.W),D0     ; Length of the name.
   Move.w    D0,-(SP)
   Move.w    #0,-(SP)         ; Cell column #.
   Move.w    D7,-(SP)         ; Cell row #.
   Move.l    NameListH(A4),-(SP)
   _LSetCell

   Addq.w    #1,D7        ; Increment the loop counter.
   Cmp.w     (A3),D7      ; Loaded all font names yet?
   Ble.s     @0           ; No -> @0.

   Select the cell containing the font name which corresponds
to the font number passed to the FontDialog Routine.  If the
font number isn't associated with one of the fonts on the
current system file then don't select any name.

   Cmpi.w    #-1,NameCell+v(A4)  ; Cell to be selected?
   Beq.s     @2                  ; No -> @2
   LMSetSelect  #True,NameCell(A4),NameListH(A4)

   DeRefHndleNameListH(A4),A0    ; Scroll if selected cell
   Move.w    visible+bottom(A0),D6  ; isn't visible, ie
                                ;the last visible cell is
   Cmp.w     NameCell+v(A4),D6   ; less than selected cell.
   Bgt.s     @2
   LMAutoScroll NameListH(A4)

@2 LMDoDraw  #True,NameListH(A4) ; Drawing on.
   Rts


LdSizes
   ; load SizeList with font sizes of TempFont in FontMap.
   LMDoDraw  #False,SizeListH(A4); Drawing off.
   LMDelRow  #0,#0,SizeListH(A4) ; Delete all the rows.
   Move.l    #$FFFF0000,SizeCell(A4)
   Move.w    TempFont(A4),D4 ;The font number.
   Bmi       @5 ; -ve, no font number passed to FontDialog.

   Clr.w     D7        ; Loop counter.
@1 Move.w    D7,D6
   Mulu      #6,D6     ; Point to the next font info.
; Compare the font number with the font number that was passed
; to the FontDialog Routine.
   Cmp.w     2(A3,D6.W),D4    ; Match?
   Beq.s     @2        ; Yes -> @2.
   Addq.w    #1,D7     ; Increment the loop counter.
   Cmp.w     (A3),D7   ; Looked at all font numbers yet?
   Ble.s     @1        ; No ->@1.
   Bra       @5        ; Yes & no match was found.

@2 Move.w    2+4(A3,D6.W),D7 ; Offset to font size list.
   Move.w    (A3,D7.W),D6    ; The number of font sizes.
   ; Procedure LAddRow (count, rowNum: Integer; lHandle:
ListHandle): Integer
   Subq.l       #2,SP
   Move.w    D6,-(SP)     ; Number of rows to add.
   Move.w    #0,-(SP)
   Move.l    SizeListH(A4),-(SP)
   _LAddRow
   Addq.l    #2,SP

   Clr.w     D6        ; Setup the loop counter.
@3 Move.w    D6,D5     ; Create an offset within size list.
   Lsl.w     #1,D5
   Add.w     D7,D5     ; Offset from the start of FontMap.
   Clr.l     D0        ; Clear .L for NumToString.
   Move.w    2(A3,D5.W),D0    ; A font size.
   Cmp.w     TempSize(A4),D0  ; Is this the selected size?
   Bne.s     @4               ; No -> @4.
   Move.w  D6,SizeCell+v(A4)  ; Store cell number to  select
@4 Lea       thename(A4),A0
   _NumToString
   ; Procedure LSetCell (dataPtr: Ptr; dataLen: Integer;
theCell: Cell; lHandle: ListHandle)
   Pea       1(A0)     ; Address of the Size (string).
   Clr.w     D0
   Move.b    (A0),D0   ; Length of the Size.
   Move.w    D0,-(SP)
   Move.w    #0,-(SP)  ; Cell column #.
   Move.w    D6,-(SP)  ; Cell row #.
   Move.l    SizeListH(A4),-(SP)
   _LSetCell

   Addq.w    #1,D6        ; Increment the loop counter.
   Cmp.w     (A3,D7.W),D6 ; Loaded all font sizes yet?
   Blt.s     @3           ; No -> @3.

   Select the cell containing the font size which corresponds
to the font size passed to the FontDialog Routine.  If the
font doesn't have the size selected, do nothing.

   Cmpi.w    #-1,SizeCell+v(A4)  ; Cell to be selected?
   Beq.s     @5                  ; No -> @5
   LMSetSelect  #True,SizeCell(A4),SizeListH(A4)

   DeRefHndleSizeListH(A4),A0
                               ; Scroll if the selected cell isn't
   Move.l    (A0),A0       ; visible, ie last visible cell is
   Move.w    visible+bottom(A0),D6  ; less than selected
cell.
   Cmp.w     SizeCell+v(A4),D6
   Bgt.s     @5
```


© The Definitive MacTutor, Vol. 4

```
   LMAutoScroll SizeListH(A4)

@5 LMDoDraw  #True,SizeListH(A4) ; Drawing on.
   Rts


SampleWindow
   ; Update the sample TextEdit Record with the currently
   ; selected font attributes.
   Btst.b    #0kActive,DFlags(A4); Are the selections valid?
   Beq       @0                  ; No -> @0
   TextFont      TempFont(A4)    ; Font number
   TextFace      TempFace(A4)    ; Font style
   TextSize      TempSize(A4)    ; Font size
   GetFontInfo  info(A4)

   DeRefHndleSample(A4),A0
   Move.w    TempFont(A4),teFont(A0)    ; teFont
   Move.b    TempFace+1(A4),teFace(A0)  ; teFace
   Move.w    TempSize(A4),teSize(A0)    ; teSize
   Move.w    info+ascent(A4),teAscent(A0)  ; teAscent
   Move.w    info+ascent(A4),D0         ; teLineHite
   Add.w     info+descent(A4),D0
   Add.w     info+leading(A4),D0
   Move.w    D0,teLineHite(A0)
   Move.l    teViewRect+topLeft(A0),teDestRect+topLeft(A0)
   Move.l    teViewRect+botRight(A0),teDestRect+botRight(A0)
   Mulu      teNLines(A0) ,D0    ; Height of the sample text.
   Move.w    SampleRect+bottom(A4),D1
   Sub.w     SampleRect+top(A4),D1
   Cmp.w     D0,D1               ; Sample greater than Rect?
   Ble       @0                  ; Yes -> @0
   Sub.w     D0,D1               ; Center the sample
   Lsr.w     #1,D1               ; lines in the middle
   Add.w     D1,teDestRect+top(A0)  ; of SampleRect.
@0 Rts


; ------------ Modal Dialog User Filter ----------
DUserFilter

   Link   A6,#0
   ; Boolean result                20(A6)
   ; Dialog (window) pointer       16(A6)
   ; Address of the event record   12(A6)
   ; Addr of word to fill in item no8(A6)
   ; Return address                4(A6)
   ; Old A6                        (A6)

   Use the event number as an index into the DEventTable.
These 12 events are all the things that could spontaneously
happen while the program is in the Modal Dialog event loop.

   Movea.l   16(A6),A4       ; Font Dialog Data Record.
   Move.w    #False,20(A6)   ; Set function result to False.
   Movea.l   12(A6),A0       ; Event Records Address.
   Move.w    evtNum(A0),D0   ; Get the event number.
   Lsl.w     #1,D0           ; *2 for table index.
   Move.w    DEventTable(D0.W),D0; Point to the routine offset
   Jmp    DEventTable(D0.W)  ; and jump to it.

DEventTable
   Dc.w   DNull-DEventTable      ; Null
   Dc.w   DMouseDown-DEventTable ; Mouse Down
   Dc.w   DFReturn-DEventTable   ; Mouse Up    (Not used)
   Dc.w   DKeyDown-DEventTable   ; Key Down
   Dc.w   DFReturn-DEventTable   ; Key Up      (Not used)
   Dc.w   DKeyDown-DEventTable   ; Auto Key
   Dc.w   DUpdate-DEventTable    ; Update
   Dc.w   DFReturn-DEventTable   ; Disk    (Not used)
   Dc.w   DActivate-DEventTable  ; Activate
   Dc.w   DFReturn-DEventTable   ; Network (Not used)
   Dc.w   DFReturn-DEventTable   ; I/O Driver (Not used)


DNull
   GetDItem  16(A6),#20,Itemtype(A4),Item(A4),ItemBox(A4)
   GetMouse  loc_pt(A4)
   PtInRect  loc_pt(A4),ItemBox(A4) ; Was mouse within the
   Move.b    (SP)+,D0               ; FontSize text edit Rect?
   Beq.s     @0                     ; Yes -> iBeam cursor.

   Move.l    TextCursor(A4),A0
   Move.l    (A0),-(SP)
   _SetCursor
   Bra       @1
@0 _InitCursor              ; No -> Arrow cursor.

@1 Move.l    Ticks,D0       ; If KeyTresh Ticks have passed
   Sub.l     SampleTicks(A4),D0  ; since last item alteration,
   Cmp.w     KeyThresh,D0 ; update the sample window by
   Blt       DFReturn       ; indicating item #21 was hit.
   Btst.b    #SUpdate,DFlags(A4)
   Beq       DFReturn

   Move.w    #21,D0         ; Set the itemNo to 21, Sample
   Bra       DFTReturn


DMouseDown
   Bclr.b    #SKey,DFlags(A4); Not a SKey event.
   ; Obtain the (events) mouse location and convert it to
   ; local coordinates.
   Move.l    evtMouse(A0),loc_pt(A4)
   GlobalToLocal loc_pt(A4)

; Check to see if the mouse was clicked inside one of the
; active rectangles.

Item9
; mouse was pressed down within item No 9? (Style heading)
   GetDItem  FontDPtr,#9,Itemtype(A4),Item(A4),ItemBox(A4)
   PtInRect  loc_pt(A4),ItemBox(A4)
   Move.b    (SP)+,D0  ; Was the point within the heading?
   Beq.s     Item14    ; No -> check item No 14's Rect.

   ; Routine TrackHeading (theDialog: DialogPtr; itemNo:
Integer): Boolean
   Subq.l    #2,SP           ; Set up for the result.
   Move.l    FontDPtr,-(SP)
   Move.w    #9,-(SP)
   Bsr       TrackHeading
   Move.b    (SP)+,D0      ; Was the mouse up within the item?
   Beq       SetoNull  ; No, Convert event to null & return

   Move.w    #9,D0     ; Yes, return True & the item No 9.
   Bra       DFTReturn

Item14
;mouse was pressed down within item No 14? (Spacing heading).
   GetDItem  FontDPtr,#14,Itemtype(A4),Item(A4),ItemBox(A4)
   PtInRect  loc_pt(A4),ItemBox(A4)
   Move.b    (SP)+,D0  ; Was the point within the heading?
   Beq       Item17    ; No -> check item No 17's Rect.

   Subq.l    #2,SP
   Move.l    FontDPtr,-(SP)
   Move.w    #14,-(SP)
   Bsr       TrackHeading
   Move.b    (SP)+,D0  ; Was the mouse up within the item?
   Beq       SetoNull  ; No, Convert event to null & return

   Move.w    #14,D0    ; Yes, return True & the item No 14.
   Bra       DFTReturn

Item17
; mouse was pressed down within item No 17?
```

```
; The Font Name Selection Window.
   GetDItem  FontDPtr,#17,Itemtype(A4),Item(A4),ItemBox(A4)
   Addi.w    #15,ItemBox+right(A4)  ; include vertical scroll
   PtInRect  loc_pt(A4),ItemBox(A4)
   Move.b    (SP)+,D0  ; Was the point within the rect?
   Beq       Item19    ; No -> check item No 19's rect.

   If a new selection is made return true and the item number,
otherwise set the event to a null event and return.

   ; Function LClick (pt: Point; modifiers: Integer; lHandle:
ListHandle): Boolean
   Subq.l    #2,SP
   Move.l    loc_pt(A4),-(SP)
   Movea.l   12(A6),A0
   Move.w    evtMeta(A0),-(SP)
   Move.l    NameListH(A4),-(SP)
   _LClick
   Addq.l    #2,SP

   Clr.l     TempCell(A4)
   LMGetSelect  #True,TempCell(A4),NameListH(A4)
   Move.b    (SP)+,D0  ; Is there a selected cell?
   Bne       @0        ; Yes -> @0.
   Move.l    #$FFFF0000,TempCell(A4); No set TempCell to 0,-
1.
@0 Move.l    TempCell(A4),D0  ; Has the selected cell changed?
   Cmp.l     NameCell(A4),D0
   Beq       SetoNull         ; No -> SetoNull.
   Move.l    D0,NameCell(A4)  ; Yes, return True & item No 17.
   Move.w    #17,D0
   Bra       DFTReturn

Item19
; Check to see if the mouse was pressed down within item No
19.
; The Font Size Selection Window.
   GetDItem  FontDPtr,#19,Itemtype(A4),Item(A4),ItemBox(A4)
   Addi.w    #15,ItemBox+right(A4)
   PtInRect  loc_pt(A4),ItemBox(A4)
   Move.b    (SP)+,D0  ; Was the point within the rect?
   Beq       DFReturn  ; No -> DFReturn.

   If a new selection is made return true and the item number,
otherwise set the event to a null event and return.

   Subq.l    #2,SP
   Move.l    loc_pt(A4),-(SP)
   Movea.l   12(A6),A0
   Move.w    evtMeta(A0),-(SP)
   Move.l    SizeListH(A4),-(SP)
   _LClick
   Addq.l    #2,SP

   Clr.l     TempCell(A4)
   LMGetSelect  #True,TempCell(A4),SizeListH(A4)
   Move.b    (SP)+,D0         ; Is there a selected cell?
   Bne       @0               ; Yes -> @0.
   Move.l    #$FFFF0000,TempCell(A4)
@0 Move.l    TempCell(A4),D0  ; Has selected cell changed?
   Cmp.l     SizeCell(A4),D0
   Beq       SetoNull         ; No -> SetoNull.
   Move.l    D0,SizeCell(A4)  ; Yes, return True & item No 19.
   Move.w    #19,D0
   Bra       DFTReturn

DKeyDown

   Examine the key that was pressed, if it was a 'cr' or
'enter' then that is equivalent to clicking the Ok button.

   Move.l    evtMessage(A0),D0
   Cmpi.b    #crCode,D0    ; Was the cr key pressed?
   Beq.s     @1            ; Yes -> @1.
   Cmpi.b    #enterCode,D0   ; Was the enter key pressed?
   Bne.s     @2            ; No -> @2.
@1 Move.w    #1,D0         ; Set the item No to Ok button.
   Bra       DFTReturn     ; Item No 1 - Ok.

@2 ;
   If the key pressed was the back space key or a number then
deselect the cell in the Font Size Selection Window (if there
is one selected), then return, allowing the number or 'bs' to
be used to alter the FontSize

   ; text edit data.
   Cmpi.b #bsCode,D0       ; Was the bs key pressed?
   Beq.s  @3               ; Yes -> @3.
   Cmpi.b #$30,D0
   Blt.s  NameSearch
   Cmpi.b #$39,D0
   Bgt.s  NameSearch
@3 Bclr.b #SKey,DFlags(A4)       ; Not a SKey event.
   Cmpi.l #$FFFF0000,SizeCell(A4); If a cell is selected,
   Beq    DFReturn                ; deselect it,
   LMSetSelect  #False,SizeCell(A4),SizeListH(A4); let the
                                  ; default routine
   Move.l #$FFFF0000,SizeCell(A4); handle the event.
   Bra    DFReturn

NameSearch
   Lea       search(A4),A2
   Bset.b    #SKey,DFlags(A4); Was last event a SKey event?
   Beq.s     @0               ; No -> @0
   Move.l    evtTicks(A0),D1  ; Yes -> If this key down
   Sub.l     SampleTicks(A4),D1 ;  within KeyThresh Ticks
   Cmp.w     KeyThresh,D1       ;  continue the search
string
   Blt.s     @1               ;  else, reset search
string.
@0 Clr.b     (A2)             ; Continue.
@1 Addq.b    #1,(A2)          ; Reset.
   Move.l    evtTicks(A0),SampleTicks(A4)
   Clr.w     D1               ; If search string is greater than
   Move.b    (A2),D1          ; 32, SetoNull.
   Cmpi.b    #31,D1
   Bgt       SetoNull
   Move.b    D0,(A2,D1.w) ; Add char to string.

   DeRefHndleFontMap(A5),A3
   Clr.w     D7               ; Set loop counter.
@2 Move.w    D7,D6
   Mulu      #6,D6            ; Offset to next font info.
   Move.w    2+2(A3,D6.w),D6  ; Offset to name string.

   ; Function IUMagString (aPtr, bPtr: Ptr; aLen, bLen:
Integer): Integer
   Subq.l    #2,SP
   Pea       1(A3,D6.w)    ; Address of FontName string.
   Pea       1(A2)         ; Address of search string.
   Clr.w     D0
   Move.b    (A3,D6.w),D0
   Move.w    D0,-(SP)      ; Length of FontName string.
   Move.b    (A2),D0
   Move.w    D0,-(SP)      ; Length of search string.
   _IUMagString
   Move.w    (SP)+,D0      ; Was the FontName => ?
   Bpl.s     @3            ; Yes -> @3

   Addq.w    #1,D7         ; Have all the fonts been
   Cmp.w     (A3),D7       ; looked at?
   Blt       @2            ; No -> @2

@3    ; Is the desired cell selected?
      ; Yes -> Don't do anything.
      ; No -> Deselect existing cell & select new cell.
```

```
    Cmp.w     NameCell+v(A4),D7
    Beq       SetoNull
    LMSetSelect  #False,NameCell(A4),NameListH(A4)
    Move.w    D7,NameCell+v(A4)
    LMSetSelect  #True,NameCell(A4),NameListH(A4)

    DeRefHndleNameListH(A4),AØ    ; If selected cell isn't
    Cmp.w     visible+top(AØ),D7  ; visible then AutoScroll.
    Blt.s     @4                  ; else continue.
    Cmp.w     visible+bottom(AØ),D7
    Blt.s     @5
@4  LMAutoScroll NameListH(A4)
@5  Move.w    #17,DØ
    Bra       DFTReturn


DUpdate
    Bclr.b       #SKey,DFlags(A4)    ; Not a SKey event.
    SetPort      FontDPtr
    BeginUpdate  FontDPtr
    DrawControls FontDPtr

; Update itemNo 1
    ; The BoldBox around the Ok button.
    GetDItem     FontDPtr,#1,Itemtype(A4),Item(A4),ItemBox(A4)
    InsetRect    ItemBox(A4),#-4,#-4
    PenSize      #3,#3
    FrameRoundRect  ItemBox(A4),#16,#16
    _PenNormal

; Update itemNo 3
    ; The Static Text heading "Character Font & Attributes".
    GetDItem  FontDPtr,#3,Itemtype(A4),Item(A4),ItemBox(A4)
    GetIText  Item(A4),thename(A4)
    Lea       thename(A4),AØ   ; Address of the text.
    Clr.l     DØ
    Move.b    (AØ)+,DØ         ; Length of the text.
    TextBox   AØ,DØ,ItemBox(A4),#Ø

; Update the Font Style Selection Window
    ; The ThinBox around the Style Check Boxes (item No 1Ø).
    GetDItem     FontDPtr,#10,Itemtype(A4),Item(A4),ItemBox(A4)
    FrameRect    ItemBox(A4)

    ; The Static Text heading "Style"    (item No 9).
    GetDItem  FontDPtr,#9,Itemtype(A4),Item(A4),ItemBox(A4)
    GetIText  Item(A4),thename(A4)
    Lea       thename(A4),AØ   ; Address of the text.
    Clr.l     DØ
    Move.b    (AØ)+,DØ         ; Length of the text.
    TextBox      AØ,DØ,ItemBox(A4),#Ø

; Update the Font Spacing Selection Window
    ; The ThinBox around the Style Check Boxes (item No 15).
    GetDItem     FontDPtr,#15,Itemtype(A4),Item(A4),ItemBox(A4)
    FrameRect    ItemBox(A4)

    ; The static text heading "Spacing"     (item No 14).

    GetDItem  FontDPtr,#14,Itemtype(A4),Item(A4),ItemBox(A4)
    GetIText  Item(A4),thename(A4)
    Lea       thename(A4),AØ      ; Address of the text.
    Clr.l     DØ
    Move.b    (AØ)+,DØ            ; Length of the text.
    TextBox   AØ,DØ,ItemBox(A4),#Ø

; Update itemNo 16
    ; The Static Text heading "Font Name:".
    GetDItem  FontDPtr,#16,Itemtype(A4),Item(A4),ItemBox(A4)
    GetIText  Item(A4),thename(A4)
    Lea       thename(A4),AØ      ; Address of the text.
    Clr.l     DØ
    Move.b    (AØ)+,DØ            ; Length of the text.
    TextBox   AØ,DØ,ItemBox(A4),#Ø

; Update itemNo 17
    ; The Font Name Selection Window.
    GetDItem     FontDPtr,#17,Itemtype(A4),Item(A4),ItemBox(A4)
    InsetRect    ItemBox(A4),#-1,#-1
    EraseRect    ItemBox(A4)
    FrameRect    ItemBox(A4)
    LMUpdate     VisRgn(A4),NameListH(A4)

; Update itemNo 18
    ; The Static Text heading "Font Size:".
    GetDItem  FontDPtr,#18,Itemtype(A4),Item(A4),ItemBox(A4)
    GetIText  Item(A4),thename(A4)
    Lea       thename(A4),AØ      ; Address of the text.
    Clr.l     DØ
    Move.b    (AØ)+,DØ            ; Length of the text.
    TextBox   AØ,DØ,ItemBox(A4),#Ø

; Update itemNo 19
    ; The Font Size Selection Window.
    GetDItem     FontDPtr,#19,Itemtype(A4),Item(A4),ItemBox(A4)
    InsetRect    ItemBox(A4),#-1,#-1
    EraseRect    ItemBox(A4)
    FrameRect    ItemBox(A4)
    LMUpdate     VisRgn(A4),SizeListH(A4)

; Update itemNo 2Ø
    ; The TextEdit text item.
    GetDItem     FontDPtr,#20,Itemtype(A4),Item(A4),ItemBox(A4)
    EraseRect    ItemBox(A4)
    TEUpdate     ItemBox(A4),teHandle(A4)
    InsetRect    ItemBox(A4),#-3,#-3
    FrameRect    ItemBox(A4)  ; Frame the textedit item.

; Update itemNo 21
    ; Update the Sample Text Window.
    EraseRect SampleRect(A4)
    Bclr.b    #SRect,DFlags(A4)
    Btst.b    #OkActive,DFlags(A4); Are selections valid?
    Beq    @Ø                     ; No -> @Ø
    TEUpdate  SampleRect(A4),Sample(A4) ; Yes -> update sample
    Move.l    Ticks,SampleTicks(A4)  ;  and associated
    Bclr.b    #SUpdate,DFlags(A4)    ;  variables.
    Bset.b    #SRect,DFlags(A4)

@Ø  EndUpdate    FontDPtr
    Bra          SetoNull

DActivate

   Check first to see if this activate event is associated
with this Dialog Window, compare the Dialog Window's pointer
to that of the Window pointerin the event record. AØ contains
the address of the event record.

    Move.l    FontDPtr,DØ
    Cmp.l     evtMessage(AØ),DØ
    Bne       DFReturn         ; Event not related to Dialog.

    Bclr.b    #SKey,DFlags(A4) ; Not a SKey event.
    SetPort   FontDPtr
    Movea.l   12(A6),AØ        ; Address of the event record.
    Move.w    evtMeta(AØ),DØ   ; Obtain the modify word.
    Lsr       #1,DØ            ; Check Bit Ø to see if its
    Bcc       DDeactivate      ; Activate or Deactivate.

    TEActivate   teHandle(A4)
    LMActivate#True,NameListH(A4)
    LMActivate#True,SizeListH(A4)
    Bra          SetoNull

DDeactivate
```

```
   TEDeactivate teHandle(A4)
   LMActivate#False,NameListH(A4)
   LMActivate#False,SizeListH(A4)
   Bra          SetoNull

SetoNull
   Movea.l   12(A6),A0     ; Obtain the address of the Event
   Clr.w     evtNum(A0)    ; Record & set it to a null event.
   Bra.s     DFReturn
DFTReturn
   Move.b    #True,20(A6) ; Dialog Filter (True) Return.
   Movea.l   8(A6),A0
   Move.w    D0,(A0)
DFReturn                   ; Dialog Filter Return.
   Unlk      A6
   Movea.l   (SP)+,A0      ; Pop args & return.
   Add.l     #12,SP
   Jmp       (A0)

TrackHeading
   Link         A6,#-22
   ; Boolean result                  14(A6)
   ; theDialog pointer               10(A6)
   ; itemNo                           8(A6)
   ; Return address                   4(A6)
   ; Old A6                           (A6)
   ; Temporary Region Handle No1     -4(A6)
   ; Temporary Region Handle No2  -8(A6)
   ; Vis boolean                   -10(A6)
   ; Mouse Location                -14(A6)
   ; Temporary Rectangle           -22(A6)

   Move.b #False,14(A6)    ; Set up to return False.

; Get Rect of user item that surrounds the headings control
; items (check boxes or radio buttons).
   Addq.w    #1,8(A6)
   GetDItem  10(A6),8(A6),Itemtype(A4),Item(A4),-22(A6)

; Create a region and store the current clipping region.

   NewRgn
   Move.l    (SP),-4(A6)  ; Store the regions handle.
   _GetClip

   Create a new clipping region which is the user item's rect
inset by 1 pixel on all sides minus the intersection rectangle
of the heading's rect and the newly created rect.

   NewRgn
   Move.l    (SP),-8(A6)     ; Store the regions handle.
   InsetRect -22(A6),#1,#1
   _OpenRgn
   FrameRect -22(A6)

   Find the rectangle that is the intersection of ItemBox(A4)
and -22(A6). If they intersect then remove that rectangle from
the clipping region.

   SectRect  ItemBox(A4),-22(A6),TempRect
   Move.b    (SP)+,D0      ; Was there an intersection?
   Beq.s     @1            ; No -> @1.
   FrameRect TempRect
@1 _CloseRgn               ; handle is already on stack.
   SetClip   -8(A6)        ; Set up the new clipping region.

   PenSize   #1,#1
   PenMode   #10           ; Inverts what ever is drawn over.

; Draw the rectangle in question, giving the user the
; impression that they have selected the heading.
   FrameRect -22(A6)
   Move.b    #True,-10(A6); Set the vis byte to True.

Track
   StillDown
   Move.b    (SP)+,D0      ; Is the mouse button still down?
   Beq       @1            ; No -> @1.
   GetMouse  -14(A6)
   PtInRect  -14(A6),ItemBox(A4)
   Move.b    (SP)+,D0      ; Has the mouse entered or
   Cmp.b     -10(A6),D0    ; left the item's rect?
   Beq.s     Track         ; No -> Track.
   FrameRect -22(A6)       ; Yes -> toggle the rectangle.
   Eori.b    #1,-10(A6)    ; Toggle the visible byte.
   Bra.s     Track

@1 Move.b    -10(A6),D0    ; Is the rect visible?
   Beq.s     @2            ; No -> Return False.
   FrameRect -22(A6)       ; Yes -> Remove the rectangle &

   Move.b    #True,14(A6) ; return True.
@2 _PenNormal
   SetClip      -4(A6)     ; Reset to its original setting.
   DisposeRgn-4(A6)    ; Dispose of two temporary regions.
   DisposeRgn-8(A6)

   Unlk      A6
   Movea.l   (SP)+,A0      ; Pop args & return.
   Add.l     #6,SP
   Jmp       (A0)

; ---------- Variables ------

FontDRec        Ds.l      1


;--------------- SetupFontMap -----------------

; Written by Ray.A.Cameron
; Version 1.0
; Mon May 4, 1988 21:41:50

XDef         SetupFontMap     ; Routine's name
XDef         FontMap          ; Global location for the
FontMap's Handle


;---------- Includes -------------

Include      Traps.D          ; Use System and ToolBox traps
Include      PackMacs.Txt     ; Use Package Equates

; ---------- Equates ----------

True         Equ    1
False        Equ    0
FNum         Equ    %1111111111000000
FSize        Equ    %0000000000111111


SetupFontMap

   Link      A6,#0
   Movem.l   D0-D7/A0-A4,-(SP)   ; Save register values.

   ; Function CountResources (theType: ResType): Integer;
   Clr.w  -(SP)         ; Clear for answer.
   Move.l #'FONT',-(SP); Count number of 'FONT' resources.
   _CountResources
   Clr.l     D7
   Move.w    (SP)+,D7  ; Store the result.

   Create a non-relocatable block into which will be stored
each font resource ID, if the resource has a font size (ie no
name).
```

```
   Move.l    D7,D0  ; Create a size to allow 2 bytes for
   Lsl.w     #1,D0  ; each resource ID.
   _NewPtr   ,Clear
   Movea.l   A0,A4  ; Store pointer to non-relocatable block.
```

Setup the Resource Manager so that the resources aren't loaded into memory.

```
   ; Procedure SetResLoad (load: Boolean);
   Move.b       #False,-(SP)     ; Load False
   _SetResLoad

   Clr.w  D3     ; Loop counter (# of font resources).
   Clr.w  D4     ; Offset into the theID non-relocatable block.
   Clr.w  D5     ; The number of different fonts.
   Clr.l  D6     ; The # of bytes to allocate to FontMap.
   ; Setup a non-relocatable block for the name of a resource.
   Move.l    #256,D0          ; The # of bytes required.
   _NewPtr   ,Clear
   Move.l    A0,thename(A5)   ; Store pointer.

ExamRes
   Addq.w    #1,D3         ; increment loop counter.
```

Obtain a handle to a resource with type 'FONT' and an index value from 1-to-CountResources.

```
   ; Function GetIndResource (theType: resType; index:
Integer): Handle;
   Clr.l     -(SP)           ; Space for handle
   Move.l    #'FONT',-(SP)   ; Font resource type
   Move.w    D3,-(SP)        ; Index value
   _GetIndResource

   ; Use the handle to obtain the resources ID and name.
   ; The handle is already on the stack.
   Pea    theID(A5)        ; point to theID
   Pea    theType(A5)      ; point to theType
   Move.l thename(A5),-(SP)   ; load address of name
   _GetResInfo

   Move.w    theID(A5),D0 ; Examine theID to see if resource
   Andi.w    #FSize,D0    ; has a font size (ie no name).
   Beq.s     NoSize       ; NoSize, ie a name-> NoSize
```

Place theID of the font resource (which contains a font size) into the non-relocatable block. Then increment the offset.

```
   Move.w    theID(A5),(A4,D4.W)
   Addq.w    #2,D4            ; increment the offset
   Bra.s     Testloop

NoSize
   Movea.l   thename(A5),A0
```

Evaluate the number of bytes to allocate to this font's name in the FontMap. Add this to the running total.

```
   Clr.l     D0
   Move.b    (A0),D0   ; Create an even length for the name
   Bset.l    #0,D0     ; (ie add padding if its rEquired).
   Addq.l    #1,D0
   Add.l     D0,D6     ; Update the running total.
   Move.l    A0,-(SP)  ;Place the pointer to the name and
   Move.w    theID(A5),-(SP) ; resource ID onto the stack.
   Addq.w    #1,D5  ; Increment the number of different fonts.
```

Create a new non-relocatable block for the next font resource name.

```
   Move.l    #256,D0   ; The # of bytes required.
   _NewPtr   ,Clear
   Move.l    A0,thename(A5)  ; Store pointer

Testloop
   Cmp.w     D3,D7     ; Have all font resources been
   Bne.s     ExamRes   ; examined, No -> ExamRes

   ; Dispose of pointer in thename(A5)
   Movea.l   thename(A5),A0
   _DisposPtr
```

Setup the Resource Manager so that the resources can be loaded into memory.

```
   ; Procedure SetResLoad (load: Boolean);
   Move.b    #True,-(SP)      ; Load True
   _SetResLoad
```

Evaluate the number of bytes required for the FontMap. Then create a relocatable block for the FontMap. D6 contains the number of bytes to allocate for the font names in the FontMap.

```
   Addq.l    #2,D6  ; 2 bytes for "the number of fonts"
   Clr.l     D0
   Move.w    D5,D0  ; The # of different fonts, each font
   Mulu      #6,D0  ; list rEquires 6 bytes.
   Add.l     D0,D6
   Move.l    D0,D3  ; Reserved for later.
```

The amount of space to allocate for the font sizes is 2 x the number of font resources. This allows one word per entry.

```
   Lsl.l     #1,D7
   Add.l     D7,D6  ; Grand Total in D6

   Move.l    D6,D0
   _NewHandle
   Move.l    A0,FontMap(A5)  ; Store the FontMap handle.
   Movea.l   A0,A2           ; Copy the handle.
   _HLock                    ; lock it

   Movea.l   (A2),A2   ; Obtain the address of the FontMap.
   Subq.w    #1,D5     ; The number of fonts- 1.
   Move.w    D5,(A2)   ; Store # of fonts- 1 in FontMap.

   Lea.l     2(A2,D3.W),A1; start of font name list.
   Lea.l     2(A2),A0      ; start of the font info list.

   ; Transfer theID and name pointer onto FontMap from stack.

Transfer
   Move.w    (SP)+,(A0)+   ; theID
   Move.l    (SP)+,(A0)+   ; name pointer.
   Cmpa.l    A0,A1  ; Have all the resources been transfered?
   Bne.s     Transfer      ; No -> Transfer.

   ; The information on the FontMap, in the font info list is
now sorted
   ; so that the first font info has the lowest (alphabeti-
cally) name
   ; associated with it.
   Move.w    (A2),D7
   Mulu      #6,D7     ; Point to last entry in list
@1 Move.w    D7,D6
@2 Subq.w    #6,D6
   Bmi.s     @3
   Movea.l   2+2(A2,D7.W),A1 ; Load pointer to aStr
   Movea.l   2+2(A2,D6.W),A0 ; Load pointer to bStr
   ; Compare the two strings.
   ; Function IUMagString (aPtr, bPtr: Ptr; aLen, bLen:
Integer): Integer;
   Clr.w     -(SP)         ; Clear for result.
```

```
   Pea      1(A1)          ; Address of aStr
   Pea      1(A0)          ; Address of bStr
   Clr.w    D0
   Move.b   (A1),D0
   Move.w   D0,-(SP)       ; Length of aStr
   Move.b   (A0),D0
   Move.w   D0,-(SP)       ; Length of bStr
   _IUMagString            ; Compare aStr and bStr
   Move.w   (SP)+,D0
   Cmpi.w   #1,D0          ; Was aStr greater than bstr
   Beq.s    @2             ; Yes -> @2.

   ; Swap the theID and the name pointers over
   Move.w   2(A2,D7.W),D0   ; Swap theIDs over
   Move.w   2(A2,D6.W),2(A2,D7.W)
   Move.w   D0,2(A2,D6.W)
   Move.l   2+2(A2,D7.W),D0 ; Swap the pointers over
   Move.l   2+2(A2,D6.W),2+2(A2,D7.W)
   Move.l   D0,2+2(A2,D6.W)
   Bra.s    @2

@3 Subq.w   #6,D7  ; Shorten the search and repeat until
   Bne.s    @1     ; the table is completely sorted.
```

Now that the entries in the font info list have been sorted the font names are loaded into the FontMap and the name pointer is now replaced by a two byte offset, As each font name is loaded its non-relocatable block is disposed of.

```
   Move.w   (A2),D5
   Addq.w   #1,D5          ; The number of fonts.
   Mulu     #6,D5
   Addq.w   #2,D5
   ; D5 contains the offset to the postion of the first font
name.

   Clr.w D7 ; loop counter (the # of fonts - 1)
LoadNames
   Move.w   D7,D6
   Mulu     #6,D6            ; Point to the next font info
   Movea.l  2+2(A2,D6.W),A3  ; Obtain pointer to the name.
   Move.w   D5,2+2(A2,D6.W)  ; Store name offset in FontMap.

   ; Perform a BlockMove to load the font's name.
   ; Procedure BlockMove (sourcePtr, destPtr: Ptr; byteCount:
Size);
   Clr.l    D0
   Move.b   (A3),D0
   Bset.l   #0,D0
   Addq.l   #1,D0     ; # of bytes required to move.
   Lea      (A3),A0   ; load source address
   Lea      (A2,D5.W),A1 ; load destination address
   Add.w    D0,D5  ; point to next name (offset)
   _BlockMove
   Movea.l  A3,A0  ; Dispose of the non-relocatable block
   _DisposPtr       ; it's no longer required.

   Addq.w   #1,D7     ; increment counter
   Cmp.w    (A2),D7
   Ble.s    LoadNames

   ; Load the font sizes
   Clr.w    D7     ; loop counter
LdFSizes
   Lea      (A2,D5.W),A0
   Clr.w    (A0)
   Move.w   D7,D6
   Mulu     #6,D6            ; Point to the next font info
   Move.w   2(A2,D6.W),D3    ; theID of the font
   Move.w   D5,4+2(A2,D6.W)  ; size list offset in FontMap
   Clr.w    D2               ; Offset into the theID block
@1 Move.w   (A4,D2.W),D0     ; Obtain the next theID
   Andi.w   #FNum,D0
   Cmp.w    D3,D0
   Bne.s    @2
   Move.w   (A4,D2.W),D0
   Andi.w   #FSize,D0
   Addq.w   #1,(A0)   ; Increment "# of font sizes" counter
   Move.w   (A0),D1   ; Offset to place font size in list
   Lsl.w    #1,D1     ; Doubled to allow for word length.
   Move.w   D0,(A0,D1.W) ; Store the font size
@2 Addq.w   #2,D2         ; Increment offset counter
   Cmp.w    D2,D4         ; Examined the complete block
   Bgt.s    @1            ; No -> @1.

   ; Sort the font size list in order of smallest to largest.
   Move.w   (A0),D3   ; The # of font sizes
   Lsl.w    #1,D3     ; Offset to the last font size
Sort1
   Move.w   D3,D2
Sort2
   Subq.w   #2,D2
   Beq.s    Sort3
   Move.w   (A0,D3.W),D1
   Cmp.w    (A0,D2.W),D1
   Bgt.s    Sort2
   Move.w   (A0,D2.W),(A0,D3.W)
   Move.w   D1,(A0,D2.W)
   Bra.s    Sort2
Sort3
   Subq.w   #2,D3
   Bne.s    Sort1

   ; Convert the font "theID" into the font number
   Move.w   2(A2,D6.W),D3
   Lsr.w    #7,D3
   Move.w   D3,2(A2,D6.W)
```

Update the offset in the FontMap (D5) to point to the location of the font sizes associated with the next font.

```
   Move.w   (A0),D0
   Addq.w   #1,D0
   Lsl.w    #1,D0
   Add.w    D0,D5

   ; Check to see if there are any more fonts to load into the
FontMap.

   Addq.w   #1,D7
   Cmp.w    (A2),D7
   Ble.s    LdFSizes
```

The FontMap has been created, and loaded with its information. Dispose of the theID non-relocatable block and unlock the FontMap.

```
   Movea.l     A4,A0
   _DisposPtr
   Movea.l     FontMap(A5),A0
   _HUnlock

   ; Restore registers to their original values
   Movem.l  (SP)+,D0-D7/A0-A4
   Unlk     A6

   Rts

; ---------- Variables ----------

theID          Ds.w     1
theType        Ds.l     1
thename        Ds.l     1
FontMap        Ds.l     1        ; Handle of the FontMap.

End
```

```
;------------------ FontMacros -------------

; Written by Ray.A.Cameron

; -----  Control Manager Routines

    ; Procedure DrawControls (gp: GrafPort)
    .Macro DrawControls
    Move.l      %1,-(SP)
    _DrawControls
    .Endm

    ; Procedure HiliteControl (theControl: Control Handle;
hiliteState: Integer)
    .Macro HiliteControl
    Move.l      %1,-(SP)
    Move.w    %2,-(SP)
    _HiliteControl
    .Endm

    ; Procedure SetCtlValue (theControl: ControlHandle;
theValue: Integer)
    .Macro SetCtlValue
    Move.l      %1,-(SP)
    Move.w    %2,-(SP)
    _SetCtlValue
    .Endm

; -----  Dialog Manager Routines

    ; Procedure CloseDialog (theDialog: DialogPtr)
    .Macro CloseDialog
    Move.l      %1,-(SP)
    _CloseDialog
    .Endm

    ; Procedure GetDItem (theDialog: DialogPtr; itemNo:Integer;
    ;   Var type: Integer; Var item: Handle; Var box: Rect)
    .Macro GetDItem
    Move.l      %1,-(SP)
    Move.w    %2,-(SP)
    Pea       %3
    Pea       %4
    Pea       %5
    _GetDItem
    .Endm

    ; Procedure GetIText (item: Handle; Var text: Str255)
    .Macro GetIText
    Move.l      %1,-(SP)
    Pea       %2
    _GetIText
    .Endm

    ; Procedure SetDItem (theDialog: DialogPtr; itemNo:
Integer; type: Integer; item: Handle; box: Rect)
    .Macro SetDItem
    Move.l      %1,-(SP)
    Move.w    %2,-(SP)
    Move.w    %3,-(SP)
    Pea       %4
    Pea       %5
    _SetDItem
    .Endm

; -----  Event Manager Routines

    ; Procedure GetMouse (Var mouseLoc: Point)
    .Macro GetMouse
    Pea       %1
    _GetMouse
    .Endm

    ; Function StillDown: Boolean
    .Macro StillDown
    Clr.w       -(SP)
    _StillDown
    .Endm

; -----  List Manager Routines

    ; Procedure LActivate (act: Boolean; lHandle: ListHandle)
    .Macro LMActivate
    Move.b      %1,-(SP)
    Move.l      %2,-(SP)
    _LActivate
    .Endm

    ; Procedure LAutoScroll (lHandle: ListHandle)
    .Macro LMAutoScroll
    Move.l      %1,-(SP)
    _LAutoScroll
    .Endm

    ; Procedure LDelRow (count, rowNum: Integer; lHandle:
ListHandle)
    .Macro LMDelRow
    Move.w    %1,-(SP)
    Move.w    %2,-(SP)
    Move.l      %3,-(SP)
    _LDelRow
    .Endm

    ; Procedure LDispose (lHandle: ListHandle)
    .Macro LMDispose
    Move.l      %1,-(SP)
    _LDispose
    .Endm

    ; Procedure LDoDraw (drawIt: Boolean; lHandle: ListHandle)
    .Macro LMDoDraw
    Move.b      %1,-(SP)
    Move.l      %2,-(SP)
    _LDoDraw
    .Endm

    ; Function LGetSelect (next: Boolean; Var theCell: Cell;
    ;         lHandle: ListHandle): Boolean
    .Macro LMGetSelect
    Clr.w     -(SP)
    Move.b    %1,-(SP)
    Pea       %2
    Move.l    %3,-(SP)
    _LGetSelect
    .Endm

    ; Procedure LSetSelect (setIt: Boolean; theCell: Cell;
lHandle: ListHandle)
    .Macro LMSetSelect
    Move.b    %1,-(SP)
    Move.l    %2,-(SP)
    Move.l    %3,-(SP)
    _LSetSelect
    .Endm

    ; Procedure LUpdate (theRgn: RgnHandle; lHandle: List-
Handle)
    .Macro LMUpdate
    Move.l      %1,-(SP)
    Move.l      %2,-(SP)
    _LUpdate
    .Endm

; -----  Memory Manager Routines
```

```
   ; Procedure DisposePtr (p: Ptr)
   ; On Entry    A0: p (pointer)
   ; On Exit     A0: 0
   ;             D0: result code (integer)
   .Macro DisposePtr
   Move.l       %1,A0
   _DisposPtr
   .Endm

   ; Procedure HLock (h: Handle)
   ; On Entry    A0 - h (Handle)
   ; On Exit     D0 - result code (integer)
   .Macro HLock
   Movea.l      %1,A0
   _HLock
   .Endm

   ; Procedure HUnLock (h: Handle);
   ; On Entry    A0 - h (Handle)
   ; On Exit     D0 - result code (integer)
   .Macro HUnLock
   Movea.l      %1,A0
   _HUnLock
   .Endm

; -----  Quick Draw Routines

   ; Procedure DisposRgn (rgn: RgnHandle)
   .Macro DisposeRgn
   Move.l       %1,-(SP)
   _DisposRgn
   .Endm

   ; Procedure EraseRect (r: Rect)
   .Macro EraseRect
   Pea      %1
   _EraseRect
   .Endm

   ; Procedure FrameRect (r: Rect)
   .Macro FrameRect
   Pea      %1
   _FrameRect
   .Endm

   ; Procedure FrameRoundRect
   .Macro FrameRoundRect
   Pea      %1
   Move.w   %2,-(SP)
   Move.w   %3,-(SP)
   _FrameRoundRect
   .Endm

   ; Procedure GetFontInfo (Var info: FontInfo)
   .Macro GetFontInfo
   Pea      %1
   _GetFontInfo
   .Endm

   ; Procedure GetPenState (Var pnState: PenState)
   .Macro GetPenState
   Pea      %1
   _GetPenState
   .Endm

   ; Procedure GlobalToLocal (Var pt: Point);
   .Macro GlobalToLocal
   Pea      %1
   _GlobalToLocal
   .Endm

   ; Procedure InsetRect (Var r: Rect; dh,dv: Integer)
   .Macro InsetRect
   Pea      %1
   Move.w   %2,-(SP)
   Move.w   %3,-(SP)
   _InsetRect
   .Endm

   ; Function NewRgn: Rgn Handle
   .Macro NewRgn
   Clr.l        -(SP)
   _NewRgn
   .Endm

   ; Procedure PenMode (mode: Integer)
   .Macro PenMode
   Move.w   %1,-(SP)
   _PenMode
   .Endm

   ; Procedure PenSize (width, height: Integer)
   .Macro PenSize
   Move.w   %1,-(SP)
   Move.w   %2,-(SP)
   _PenSize
   .Endm

   ; Function PtInRect (pt: Point; r: rect): Boolean
   .Macro PtInRect
   Clr.w    -(SP)
   Move.l   %1,-(SP)
   Pea      %2
   _PtInRect
   .Endm

   ; Function SectRect (srcRectA, srcRectB: Rect; Var dstRect:
Rect): Boolean
   .Macro SectRect
   Clr.w    -(SP)
   Pea      %1
   Pea      %2
   Pea      %3
   _SectRect
   .Endm

   ; Procedure SetClip (rgn: RgnHandle)
   .Macro SetClip
   Move.l   %1,-(SP)
   _SetClip
   .Endm

   ; Procedure SetPenState (pnState: PenState)
   .Macro SetPenState
   Pea      %1
   _SetPenState
   .Endm

   ; Procedure SetPort (gp: GrafPort)
   .Macro SetPort
   Move.l   %1,-(SP)
   _SetPort
   .Endm

   ; Procedure TextFace (face: stlye)
   .Macro TextFace
   Move.w   %1,-(SP)
   _TextFace
   .Endm

   ; Procedure TextFont (font: Integer)
   .Macro TextFont
   Move.w   %1,-(SP)
   _TextFont
   .Endm
```


© The Definitive MacTutor, Vol. 4

```
    ; Procedure TextSize (size: Integer)
    .Macro TextSize
    Move.w    %1,-(SP)
    _TextSize
    .Endm

; -----  Resource Manager Routines

    ; Function GetResource (theType: ResType; theID: Integer):
Handle
    .Macro GetResource
    Clr.l     -(SP)
    Move.l    #%1,-(SP)
    Move.w    %2,-(SP)
    _GetResource
    .Endm

; -----  Text Edit Routines

    ; Procedure TEActivate (hTE: TEHandle)
    .Macro TEActivate
    Move.l    %1,-(SP)
    _TEActivate
    .Endm

    ; Procedure TEDeactivate (hTE: TEHandle)
    .Macro TEDeactivate
    Move.l    %1,-(SP)
    _TEDeactivate
    .Endm

    ; Procedure TEDispose (hTE: TEHandle)
    .Macro TEDispose
    Move.l    %1,-(SP)
    _TEDispose
    .Endm

    ; Procedure TEUpdate (rUpdate: Rect; hTE: TEHandle)
    .Macro TEUpdate
    Pea       %1
    Move.l    %2,-(SP)
    _TEUpdate
    .Endm

    ; Procedure TextBox (text: Ptr; length: LongInt; box :Rect;
    ;         just: Integer)
    .Macro TextBox
    Move.l    %1,-(SP)
    Move.l    %2,-(SP)
    Pea       %3
    Move.w    %4,-(SP)
    _TextBox
    .Endm

; -----  Window Manager Routines

    ; Procedure BeginUpdate (gp: GrafPort)
    .Macro BeginUpdate
    Move.l    %1,-(SP)
    _BeginUpdate
    .Endm

    ; Procedure EndUpdate (WindowPtr: WindowPtr)
    .Macro EndUpdate
    Move.l    %1,-(SP)
    _EndUpdate
    .Endm

    ; Procedure HideWindow (theWindow: WindowPtr)
    .Macro HideWindow
    Move.l    %1,-(SP)
    _HideWindow
    .Endm

    ; Procedure ShowWindow (theWindow: WindowPtr)
    .Macro ShowWindow
    Move.l    %1,-(SP)
    _ShowWindow
    .Endm

; -----  Miscellaneous Routines

    .Macro DeRefHndle
    Movea.l   %1,%2
    Movea.l   (%2),%2
    .Endm

; File Window.Link

!Start

/Output Window

[

FontMap
FontDialog
Window

/Include Window.Rsrc

$


* Window.R
* resource file for the program called "Window"
*

Window.Rsrc
????????

Type RACA = STR
   ,0
● by Ray A. Cameron of Australia \0Dver 4 MAR 1988

Type FREF
,128
APPL 0
,129
TEXT 1

Type BNDL
,128
RACA 0
ICN#
0 128 1 129
FREF
0 128 1 129

* ------ Multifinder events ---------

* bit 15 = switcher save screen
* bit 14 = accept suspend resume events
* bit 13 = switcher enable option switch
* bit 12 = can do background on null events
* bit 11 = multifinder aware
*         (activates & deactivates topmost
*           window at resume, suspend events)

Type SIZE = GNRL
   ,-1
.H
4800      ;; $4800 = bits 14,11 set
.L
128000 ;; (for 150K recomended)
.L
```

```
80000  ;; (for 80K minimum)
.I

*
* MENU Resource #1 specifies the menus used by the Window
program.
* For proper support of the Desk accessories, the Apple menu
* should be first, and the Edit menu should be third.  The
first 5 items
* in the Edit menu should be identical to those used below.
This makes
* it possible for the desk accessories to share the Edit menu
with your
* application.
*

Type MENU
  ,1 (16)
\14
 About This Example...
 (-

  ,2 (16)
File
  Quit/Q

  ,3 (16)
Edit
  (Undo/Z
  (-
  Cut/X
  Copy/C
  Paste/V
  Clear

  ,4 (16)
Font
  Dialog…/F

* Dialog Resource #1 specifies properties of the About box.
It points
* to Dialog Item List (DITL) Resource #1 as containing its
items.

Type DLOG
  ,1

100 100 190 400
Visible  NoGoAway
1
0
1

* Dialog Resource #260 specifies properties of the Font  box.
It points
* to Dialog Item List (DITL) Resource #260 as containing its
items.

   ,260
This is the Font Modal Dialog.
30 38 330 473
Invisible  NoGoAway
1
0
260

* Alert Resource #260 specifies properties of the Font Alert
box.  It points
* to Dialog Item List (DITL) Resource #261 as containing its
items.

Type ALRT
   ,260
30 38 100 473
261
6665

* Dialog Item List Resource #1 specifies the items in the
About box.
* By convention, the first item in an item list is the OK
button.
* If there is a cancel button, it should be second.  This
makes it
* easier to interpret the item number returned by the call to
ModalDialog.

Type DITL
  ,1
3

Button
60 230 80 290
OK

StaticText
15 20 36 300
This sample program was written

StaticText
35 20 56 300
just to prove it could be done!

* Dialog Item List Resource #260 specifies the items in the
Font box.
   ,260
21

button
8 359 26 427
Ok

button
8 281 26 349
Cancel

staticText Disabled
4 4 18 194
Character Font & Attributes

checkBox
63 339 79 422
Bold

checkBox
79 339 95 422
Italic

checkBox
95 339 111 422
Underline

checkBox
111 339 127 422
Outline

checkBox
127 339 143 422
Shadow

staticText
45 339 61 375
Style

userItem Disabled
53 331 151 427
```


© The Definitive MacTutor, Vol. 4

```
radiobutton
95 235 111 318
Normal

radiobutton
111 235 127 318
Condense

radiobutton
127 235 143 318
Extend

staticText
77 235 93 288
Spacing

userItem Disabled
85 227 151 323

staticText
34 4 48 81
Font Name:

userItem
54 5 150 126

staticText
34 151 48 218
Font Size:

userItem
54 152 150 186

editText
34 226 48 261

userItem Disabled
158 4 293 427


   ,261
2
```

```
button
8 359 26 427
Ok

staticText Disabled
25 91 61 328
Please select a font ^0^1^2


* WIND Resource #1 specifies the title, coordinates, and other
status
* for the window in which editing takes place.  It is dis-
played by a
* call to GetNewWindow.

Type WIND
   ,1
A Sample
50 40 300 450
Visible NoGoAway
0
0


* STR  Resource #261 to #263 are the strings associated with
* the Font Alert box. Informing the user as to the invalid
information.
* Resource #264 is the sample text displayed at the bottom of
the
* Font Modal Dialog Box.

Type STR
   ,261
name
   ,262
size between 4 and 127
   ,263
and a font
   ,264
ABCDEFGHIJKLMNOPQRSTUVWXYZ\0D++
abcdefghijklmnopqrstuvwxyz\0D++
1234567890 \0D++
!@#$%^&*()
```

Assembly Lab
Faster Bitmap Rotation

Assembler

Mike Morton
Contributing Editor
Pearl City, HI
MacTutor Vol. 4 No. 11

Faster Bitmap Rotation Optimizing By Tuning Assembler Code

Mike Morton, University of Hawaii

In last February's MacTutor, John Olsen showed a routine which copied one bitmap to another — and rotated it 90°. For speed, John coded his rotation routine in assembler.

Maybe it's the effect of working in 68000 assembler for the last couple of years, or maybe I'm just being ornery, but I noticed a few things I wanted to try to speed up. John put it best when he said "Make your code do the bare minimum first, then, after it is working, enhance it."

In this article we'll try to enhance the speed of John's rotation routine. If bitmap rotation isn't your cup of tea, you may want to read along anyway — the lessons of tight assembler code are applicable to many other problems.

Also, if you didn't follow every detail of John's writeup, don't worry — I didn't either. One approach to optimizing code is to look at what it does, not what it's trying to do, then change it to do the same thing faster. Most of the optimizations I found could have been found even if I didn't know exactly what the routine was doing.

Getting a Baseline

To start with, let's do some preliminary cleanup and find out just how fast the routine is. There are a couple of NOP instructions in the inner loop of the on-disk copy (they're not in the magazine version; I'm not sure why...) which we can ditch.

Then there are some unused labels. They're an interesting way to document points in assembler code, but I think they clutter things up when you're looking for speed. It's very important to know what the flow of control is. The removable labels are: computeInitLowerLeft, doneWithSrcWord, doneWithRow, doneWithDestWord, and cleanUpTime.

Now let's put in some timing. We won't be concerned with perfect accuracy, so the "tick count" will do fine. We'll add a local variable called "startTime" to the other local variables. Then, at the beginning of the routine, add:

```
move.l ticks, startTime-localVars(a0)
```

to strobe the starting time. At the end, just before exiting, add:

```
move.l  ticks, d0
sub.l   startTime-localVars(a0), d0
_Debugger
```

This gets the final time, computes the elapsed time, and drops into the debugger so I can see the elapsed time in register D0.

An easier way would be to compute the time the same way, but to return it to the Pascal program. I don't have the MPW

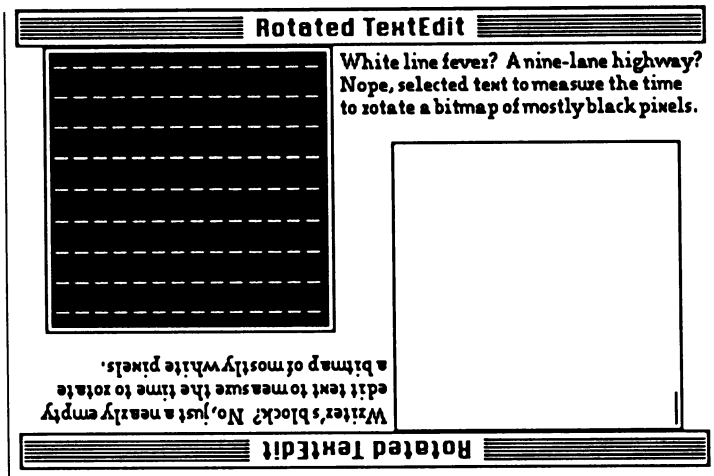


Figure 1.

Pascal compiler, so I had to keep the timing at the assembler level. It's a drag being constantly forced into the debugger, so when actually using this program, I'd often patch the _Debugger trap to a NOP, and then set a breakpoint there when I was ready to do the timing.

The tick count is accurate to only 1/60th of a second, and the difference between two successive measurements is twice as inaccurate. To avoid part of this, you can synchronize the starting measurement with the beginning of one tick:

```
        move.l  ticks, d0       ; get the time
@1      cmp.l   ticks, d0       ; changed yet?
        beq.s   @1              ; nope: loop
```

When you actually run the program, adding this code makes measurements noticeably more consistent.

The Envelope Please...

When you run the program with the above changes, how fast is it? Well, it depends on what kind of image you're copying: white pixels copy almost twice as fast as black ones. This is because the code sets the target bitmap to all white and then copies any black pixels in the source bitmap.

As an initial benchmark, the four-way rotation program takes 12 or 22 ticks, for white and black pixels, respectively. The white-pixel benchmark is when the edit text is cleared completely. The black-pixel case is when the edit text is all "—"s, selected to highlight it. (See figure 1.)

So, from 12 and 22 ticks, how much can we improve?

Start In The Middle!

The central loop is where you should always start looking. It starts out by testing a bit in the word from the source bitmap:

```
btst    d6, d3
```


© The Definitive MacTutor, Vol. 4

```
beq     NoBitToChange
```

Since D6 runs from 15 down to 0 during the loop, it checks each of the bits, going left-to-right. Instead of an explicit bit test, we can shift these bits off, one at a time. The fast way to shift a register left by a bit is to add it to itself. So discarding a bit and testing the carry can be done with:

```
add.w   d3, d3          ; test next bit…
bcc.s   NoBitToChange   ; …branch if not set
```

Is this all that important? Well, not really by itself. (But it will help us get rid of D6 a little later on.) The speed with this change goes from (12, 22) to (12, 20). I think the ADD.W isn't actually faster, but the short branch is faster than the long branch for the black-pixel case.

That's The Carry; Now The Cache

Another rule of thumb is to keep variables in registers. Let's look a little farther ahead in the same central loop:

```
movea.l currentWord-localVars(a0), a2
```

This loads a value which doesn't actually change during the loop. In fact, we can get rid of the "currentWord" variable entirely, and just use A2 in its place. Then we can delete the line above, since the value is "cached", and alter all three references to "currentWord-localVars(a0)" to just use "A2". Note that one of the lines becomes:

```
sub.l   d7, a2
```

It doesn't matter if this is a long or word subtract — they're the same for an address register operation.

This speeds up the benchmark surprisingly, to (8, 12) ticks. Even the white-pixel case improves, because A2 is subtracted from in both cases. These two simple changes have given us a speed improvement of between 33% and 45%!

Actually, the second timing for this version sometimes pops up as 13, not 12 ticks. This is nice, because even the slighest improvement will now change it to 12. For instance...

Get The Address Right

Addressing modes are important, too. For instance, just before the inner loop, the code picks up the next source word with:

```
move.w  (a1), d3
```

and after the loop, it advances to the next word with:

```
addq.l  #2, a1
```

Instead of this, you can delete the second line and change the first line to:

```
move.w  (a1)+, d3
```

This little improvement is barely measurable. It just stabilizes the second measurement at 12 ticks.

Overbyte or Underbyte?

Now, one of the remaining big pieces of the inner loop is:

```
move.w  (a2),d5
bset    d4,d5
move.w  d5,(a2)
```

The idea here is that A2 points to a word in memory. But BSET acting on memory operates only on a byte. The existing code has to load a register so it can index up to 16 bits in the word.

When D4 is between 15 and 8, a bit in the top byte of the word is set. When D4 is between 7 and 0, a bit in the low byte is set. So if you knew that D4 acted on the high byte, you could use:

```
bset    d4, (a2)  ; set it in high byte
```

And if you knew it acted on the low byte, you could use:

```
bset    d4, 1(a2) ; set it in low byte
```

Note that the bit number is taken modulo eight, so D4 needs no adjustment for the former case (no need to subtract eight).

But it's too expensive to make a check in the middle of the loop. Luckily, D4 stays the same over a stretch larger than the loop. So we can just adjust A2 to point to the right byte. Before the "rowloop" label, the new code is:

```
cmp.w   #7, d4  ; doing high byte or low?
bgt     dobyte1 ; skip if high byte
addq    #1, a2  ; else point to low byte
dobyte1
```

And after the end of the loop, after "bne rowloop", this is compensated for with:

```
cmp.w   #7, d4  ; did we do high or low?
bgt     dobyte2 ; skip if high byte
subq    #1, a2  ; else undo fudge to a2
dobyte2
```

The reward for all this effort is that those three instructions to load, alter, and store a word become this simple one:

```
bset    d4, (a2) ; set bit in high or low
```

This doesn't help the white-pixel case, since the code above is used only to copy a black pixel. But the copy case speeds up from 12 ticks to 10.

Note that using OR.W to set the bit would have been slower than the bit-setting method. Now, for the final optimization:

Roll Over, John Olsen

D6 isn't used as anything except a counter in the inner loop, which now looks like:

```
add.w   d3, d3          ; test bit in source
bcc.s   NoBitToChange   ; not set? skip
bset    d4, (a2)        ; set in high/low
noBitToChange
sub.l   d7, a2          ; subtract rowBytes
dbra    d6,innerLoop    ; loop 'til d6 = -1
```

This is 5 instructions, one of which is devoted solely to counting a constant sixteen iterations. This is easy to fix: just repeat the first 4 instructions in-line sixteen times, and forget about counting at runtime. Except for the 16 labels to type in, a quick copy-and-paste session does this neatly. Plus you can ditch the code which initializes D6:

```
moveq.l #0,d6   ; clear d6
move.w  #15,d6  ; init inside loop counter
```

This technique, "unrolling the loop", is a big win! Both timings drop by 3 ticks for a final result of (5, 7) — down from an original benchmark of (12, 22). As a wag on Usenet remarked the other day "Never put off 'til runtime what you can do at compile-time."

The code printed along with this is the second half of the original ROTATE.A, with the above changes (there were no changes in the first half, which isn't time-critical).

A Hack Too Far

There's probably more that could be done here. For example, instead of advancing the local variable "wordCount" up

to a limit, decrementing it down to zero is faster to test. You can always squeeze another percent out of any piece of code — but is it worth it? The routine published here is "optimum", in my opinion; it's been improved significantly without making the code unrecognizable or too unreadable. If this routine were actually used in a product, you might want to add (or even remove) some of these optimizations, depending on how important speed was.

Summary

With a couple of evenings of work, the bitmap-rotation routine's performance has been improved by a factor of about 2.5 to 3. Standard rules of thumb were what helped the most:

- instrument precisely
- concentrate on the inner loop
- make efficient use of registers
- exploit addressing modes for the instruction set
- unroll important loops

Almost any significant piece of code, in any language, can benefit by a review by someone other than the author. The original author "knows" too well what the code is supposed to do. One approach for a programmer trying to optimize is to look at the code without comments, just look at what's happening to the data. Being forced to reverse-engineer a subroutine can help inspire "peephole" optimizations. The Zen of Low-level Optimization consists of looking at what the code does, not what it was intended to do.

Listing: rotate.a

```
   INCLUDE  'SysEqu.a'   ;the usual system equates & macros

   INCLUDE  'Traps.a'
   INCLUDE  'QuickEqu.a'

newPtrClear FUNC EXPORT

   IMPORT SAVERETA1

   MOVE.L (SP)+,A1
   MOVE.L (SP)+,D0
   _NewPtr  clear
   MOVE.L A0,(SP)
   JMP    SAVERETA1

   ENDF


Rotate PROC EXPORT
;_____________________________________________________
;
; PROCEDURERotate (sourceMap, destMap : bitMap );
;
; given a source and destination bitmap rotate it CCW 90°…
; bitmap can be of any size, we will align on word boundaries
; (v & h)
;_____________________________________________________
; a0 = result of _newPtr call  | d0 = size of bitMap allocate
; a1 = ptr to src bitMap       | d1 = width/2  (source)
; a2 = ptr to dest bitMap      | d2 = height/2 (source)
; a3 = dest bits               | d3 = left | top
; a4 = copy of dest bits       | d4 = ctrPt.h
; a5 = not used                | d5 = ctrPt.v
; a6 = not used                | d6 = right | bottom
; a7 = not used                | d7 = rowBytes of dest bitMap
;_____________________________________________________
;
;*************************************************************
; *                                                          *
; *      Initialize destination bitMap (destMap), etc…       *
; *                                                          *
;*************************************************************
; parameter offsets
;
DestMap    EQU          4+4 ; offset to dest bitmap ^
SourceMap  EQU DestMap + 4  ; offset to source bitmap ^
;
; create local stack frame, and save some registers
; make room for dest rect to be passed back
;
  link   a6,#0              ; set up a stack frame (no bytes)
  movem.l  d3-d7/a2-a5,-(sp) ; save registers pascal may need

  move.l SourceMap(a6),a1  ; point to the source BitMap
  move.l DestMap(a6),a2    ; point to the dest BitMap
;
; compute height of source BitMap rounded up to nearest word
;
  move.w bounds+bottom(a1),d7; get bottom and put in d7
  sub.w  bounds+top(a1),d7 ; bottom - top = height, put in d7
  move.w d7,d0             ; save a copy of height for later
  ; rnd = (height+15)/16, follows:
  addi.w #15,d7            ; add 15 to the height, put in d7
  lsr      #4,d7           ; divide by 16
  lsl      #1,d7           ; multiply by 2 = rowBytes of dest
bitMap
;
; d7 now contains rowBytes of dest bitMap
; now compute the destRect for dest bitMap
;
; compute center point of Source rect
; note the center points are not necess. the
; same in the vert dir. relative to srcMap
; because of rounding that follows (i.e. height
; of srcMap is dir. prop. to RowBytes of destMap)
  lsr      #1,d0           ; divide height by 2
  move.w d0,d2             ; make a copy of height/2 for later
  move.w bounds+top(a1),d5 ; get 'top' of source rect
  add.w  d0,d5             ; d5 contains vert compnt of ctr pt
  ; compute the width of src rect & put in d4
  move.w bounds+right(a1),d3 ; get right and put in d3
  sub.w  bounds+left(a1),d3  ; right - left = width, put in d3
  lsr      #1,d3           ; divide width by 2
  move.w d3,d1             ; make a copy of width/2 for
later
  move.w bounds+right(a1),d4 ; get right of source rect
  sub.w  d3,d4             ; d4 contains horiz compnt ctr pt
;
; now get center point and compute bounds for dest bitMap
; height and width are reversed now for the 2 bitMaps
; left   :=ctrPt.h - height/2
; top    :=ctrPt.v - width/2
; right  :=ctrPt.h + height/2
; bottom:=ctrPt.v + width/2
; SetRect(rect, left, top, right, bottom) put data direct to
; avoid the overhead of a trap call
;
; compute 'top'
  move.w d5,d3          ; get a copy of ctrPt.v and put in d3
  sub.w  d1,d3          ; ctrPt.v - width/2 = 'top' in d3
  swap   d3             ; move 'top' to hiword
                        ; compute 'left'
  move.w d4,d3; get a copy of ctrPt.h and put in loword d3
  sub.w  d2,d3          ; ctrPt.h - height/2 = 'left' in
loword d3
;
; we now have 'top' in hiword & 'left' in loword d3
```


© The Definitive MacTutor, Vol. 4

```
;
                        ; compute 'bottom'
  move.w d5,d6          ; get a copy of ctrPt.v in of d3
  add.w  d1,d6          ; ctrPt.v + width/2 = 'bottom' in d3
  swap   d6             ; move 'bottom' to hiword
                        ; compute 'right'
  move.w d4,d6; get a copy of ctrPt.h and put in loword d6
  add.w  d2,d6; ctrPt.h + height/2 = 'right' in loword d6
;
; we now have 'bottom' in hiword & 'right' in loword d6
;
; assign data to dest bitMap structure directly
;
  move.w d7,rowBytes(a2) ; put in rowBytes for dest bitMap
  move.l d3,bounds+topLeft(a2) ; put the coord pair in direct
  move.l d6,bounds+botRight(a2); put the coord pair in direct
  move.w d7,d0                 ; put rowbytes in d0
  mulu   rowBytes(a1),d0   ; multiply rowBytes src*dest = d0
  lsl      #3,d0; multiply d0 by 8 for size of dest bitMap
  _NewPtr  clear; allocate size of dest bitMap (a2 pts to it)
  move.l a0,baseaddr(a2)   ; move a0 to baseAddr field
                           ; of dest bitMap structure

;_____________________________
; a0 = baseAddr for our locals   | d0 = rowbytes of srcMap
; a1 = points to bits of src bitMap| d1 = width/2 (source)
; a2 = ptr to dest bitMap        | d2 = height/2(source)
; a3 = dest bits                 | d3 = current word srcMap
; a4 =                           | d4 = copy of width of srcMap
; a5 = not used                  | d5 = current destMap word
; a6 = not used           | d6 = insideLoop count (current bit)
; a7 = not used           | d7 = rowBytes of dest bitMap
;_____________________________
; ************************************************************
; *
; *      Start actual rotation of bitMaps
; *
; ************************************************************
; Assumptions:
; height of destination bitMap = rowBytes of srcMap
;
;
; Get the baseAddr for our locals into a0 - this will make our
; program much more readable as we will let the assembler do
; all the addr location calcs for us. The addresses will be a
; fixed number while the programs is (locked down) and
running,
; therefore, there is no runtime penalty for doing this addr
;calculation

  lea    localVars,a0

  ; Synchronize with the TickCount rollover.
  move.l ticks, d0          ; get the time
@1cmp.l  ticks, d0          ; changed yet?
  beq.s  @1                 ; nope: loop

  ; Strobe the starting time.
  move.l ticks, startTime-localVars(a0)
; save starting time, in ticks
;
; initialize d0 to contain the pointer to the srcMap
;
  move.w rowbytes(a1),d0   ; get rowbytes of srcMap
; get regs pointing to bits
;
  move.l (a1),a1           ; a1 pts to BITS of src bitMap

  move.l (a2),a3               ; a3 pts to BITS of dest bitMap
;
; compute the lower left corner of the destMap - this maps to
; the upper left corner of the srcMap and will be used as base
; from which to offset intothe dest map when we need to set a
; bit
  clr.l  d1             ; clear d1
  move.w d0,d1          ; load rowBytes of srcMap
                        ; (rowBytes of srcMap = ht of destMap)
  lsl.w  #3,d1; multiply by 8 to get ht of destMap when rot'd
  mulu.w d7,d1; multiply by rowbytes(destMap) = #words in
  sub.w  d7,d1                ; destMap less one row
  add.l  a3,d1
; add the computed offset to the baseAddr of destMap
  move.l d1,lowerLeft-localVars(a0) ; init lowerLeft
;
;_____________________________
; a0 = baseAddr for our locals   | d0 = rowbytes of srcMap
; a1 = points to bits of src bitMap | d1 = SCRATCH
; a2 = SCRATCH           | d2 = colmCounter
; a3 = dest bits         | d3 = current srcMap word
; a4 = SCRATCH           | d4 = colmBitLoop counter
; a5 = not used          | d5 = current word destMap
; a6 = not used          | d6 = insideLoop count (current bit)
; a7 = not used          | d7 = rowBytes of dest bitMap
;_____________________________
;
  _Debugger;
; init wordCount to zero
;
  move.w #0,wordCount-localVars(a0) ; set wordCount to one
before we start
;
; init colmLoop counter
;
  move.w d7,d2                  ; get rowBytes(dest)
  lsr      #1,d2
; divide by 2 - d2 now contains the outer counter
  move.w d2,colmCount-localVars(a0)
; put the max value away for reference
  moveq.l  #0,d2
; set d2 to zero since we use it for the counter
colmLoop
;
; this is the outside loop
;
  move.w #15,d4              ; init colmBitLoop counter to 15
  move.l lowerLeft-localVars(a0), a2  ; init currentWord

colmBitLoop
; this is the loop that does the bit counting in the destMap -
; it is really a misuse of the 'dbra' instruction since it is
; not an independent loop but itis a quite cheap method of
; doing our counting

  cmp.w  #7, d4          ; are we doing high byte or low?
  bgt      dobyte1       ; skip if high byte
  addq   #1, a2          ; else point to low byte
dobyte1

rowLoop
;
; loop across each row in the srcMap and keep count each time
; we do a row this 'rowCount' will used to compute the offset
; from 'lowerLeft'
  move.w (a1)+, d3       ; get a word to rotate into d3
  addq.w #2,wordCount-localVars(a0) ; add 1 for each word
ooojdoooo

  add.w  d3, d3; test next bit in the current source word
  bcc.s  NoBitToChange       ; if bit's not set, skip to next
  bset   d4, (a2)
; set proper bit in high or low byte in currentWord
noBitToChange
  sub.l  d7, a2
; subtract rowBytes(destMap) from current word

  ; Do the same as above, fifteen more times.
  add.w  d3, d3
  bcc.s  @1
```

```
  bset   d4, (a2)
@1sub.l  d7, a2

  add.w  d3, d3
  bcc.s  @2
  bset   d4, (a2)
@2sub.l  d7, a2

  add.w  d3, d3
  bcc.s  @3
  bset   d4, (a2)
@3sub.l  d7, a2

  add.w  d3, d3
  bcc.s  @4
  bset   d4, (a2)
@4sub.l  d7, a2

  add.w  d3, d3
  bcc.s  @5
  bset   d4, (a2)
@5sub.l  d7, a2

  add.w  d3, d3
  bcc.s  @6
  bset   d4, (a2)
@6sub.l  d7, a2

  add.w  d3, d3
  bcc.s  @7
  bset   d4, (a2)
@7sub.l  d7, a2

  add.w  d3, d3
  bcc.s  @8
  bset   d4, (a2)
@8sub.l  d7, a2

  add.w  d3, d3
  bcc.s  @9
  bset   d4, (a2)
@9sub.l  d7, a2

  add.w  d3, d3
  bcc.s  @10
  bset   d4, (a2)
@10 sub.l  d7, a2

  add.w  d3, d3
  bcc.s  @11
  bset   d4, (a2)
@11 sub.l  d7, a2

  add.w  d3, d3
  bcc.s  @12
  bset   d4, (a2)
@12 sub.l  d7, a2

  add.w  d3, d3
  bcc.s  @13
  bset   d4, (a2)
@13 sub.l  d7, a2

  add.w  d3, d3
  bcc.s  @14
  bset   d4, (a2)
@14 sub.l  d7, a2

  add.w  d3, d3
  bcc.s  @15
  bset   d4, (a2)
@15 sub.l  d7, a2
;
; at this point we've just completed a word from the srcMap
;
  cmp      wordCount-localVars(a0),d0
; cmp wordCount to rowBytes-are we done with a row ?
  bne      rowLoop             ; no - do another word

  cmp.w  #7, d4                ; did we do high byte or low?
  bgt      dobyte2             ; skip if high byte
  subq   #1, a2                ; else undo the fudge to a2
dobyte2

;
; we just completed a row so add 2 to the count
;
  move.w #0,wordCount-localVars(a0)
; 000jd0000 reset wordCount to zero

;
; the following use of dbra keeps track of destMap bit for us
;
  move.l lowerLeft-localVars(a0), a2  ; re-init currentWord

          ; set the currentWord to the lowerLeft(current)
          ; lowerLeft is moved over (right) by a word each
          ; time we do a row in the srcMap
  dbra   d4,colmBitLoop       ; go thru the bits (15->0) in the
                              ; destMap - then exit and do it
                              ; again for rowBytes/2 times

;
; were done counting thru the bits of a destMap word - at this
; point we have actually finished a column in the destMap. The
; column is a multiple of 16 bits wide (and being at least 1
; colm) and as tall as the destMap is (which is also equal to
; rowBytes of the srcMap)
  addq.l #2,lowerLeft-localVars(a0)
; this moves us into the next
; 'column' of bits in the destMap
  addq.w #1,d2                ; add 1 to the colmLoop counter
  cmp      colmCount-localVars(a0),d2   ; are we done yet ?
  bne      colmLoop                ; no - do it again
                                   ; else we're done
;
; were done so lets do our clean-up/restore and go away
; gracefully
  move.l ticks, d0            ; get the current time…
  sub.l  startTime-localVars(a0), d0 ; …less starting time
  _Debugger                   ; and let me see the time it took

  movem.l  (sp)+,d3-d7/a2-a5 ; restore registers
  unlk   a6                   ; clean up stack frame
  move.l (sp)+,a0             ; return address
  add.l  #8,sp                  ; pop parameters off stack
  jmp      (a0)                 ; bye y'all!
; set up our data storage area
;
localVars    ; the beginning of our local var storage area
colmCount    ds.w       1
currentWord  ds.l       1
lowerLeft    ds.l       1
wordCount    ds.w       1
startTime    ds.l       1

  ENDP
  END
```

Listing: Build Script

```
### Assemble, link, and rez the rot4edit program,
### including the changes in rotate.a.
asm rotate.a -o rotate.a.o -wb
link -o testProg rotate.a.o rot4edit.p.o
(mpw)libraries:Runtime.o (mpw)libraries:Interface.o
(mpw)Plibraries:paslib.o
rez -a -o testProg rot4edit.r
testProg
```

Åssembly Lab

Smoothing via PicComments

Raul Tabasso
Rome, Italy
MacTutor Vol. 4 No. 12

Inside PicComments

Everything you've always wanted to know about picComments but were afraid to ask.

Raul Tabasso is currently working on his thesis in Architecture at the University of Rome where he was particularly involved in designing a CAD system. At the end of 1984 he started developing software on the Mac and since then it has become his principle activity. At the moment he is working on a large project, a Macintosh application written in assembly language that will eventually allow bitmaps (like those digitized with scanners) to be transformed into a collection of objects as in a PICT file. In his younger days Raul worked for a week as a bicycle messenger in San Francisco where he learned how to overcome slopes.

When I first read the *Laser Manual* and *Apple Technical Notes* on the subject, I was very surprised to see how easy it seemed to implement some laser specific features without the bother of sending a special PostScript (PS) program to the printer. Actually the idea of passing information to the laser driver through *picture comments* not only makes it very easy to include some features in your programs, but guarantees longevity and stability to your code utilizing the power of PS through QuickDraw (QD). In fact even if one day this technique could be replaced, Apple will hardly change the rules its own applications and many others depend on. This is at least if you don't take *picComments* (from now on *picComs*) just like a trick to send your own PS programs which, although it now works, could really break the rules of device independence.

If you have experienced *picComs* you probably realized that, despite revisions, Apple's own documentation surprisingly lacks precision and clearness in more than one point and that such a brilliant idea appears not to have been supported as it should. Since a lot has been already written about *picComs* I am not going to repeat basic information most of us already read but I will concentrate on the more obscure and less sampled aspects. For example one of the most exciting features of *picComs* is the ability to have easy access to PS equipped printer's capability to smooth polygons using *bezier* curves. As far as I know (you can check how far in the bibliography), nothing has ever been shown about printing smoothed polygons using this scheme. Furthermore we will examine the inner working of one more 'unrevealed' *picCom*, the *SetLineWidth*.

• Please note that through this article we will use the term laser or laser driver for convenience where we shall actually be referring to any device and driver that can process *picComs*. Seemingly the term imagewriter will be referred to those devices and drivers ignoring *picComs*. Moreover, since there is no reason your application can't make sure it is running on a recent driver, I cannot guarantee everything will work as described in this article using earlier versions of the 4.0 laser driver •

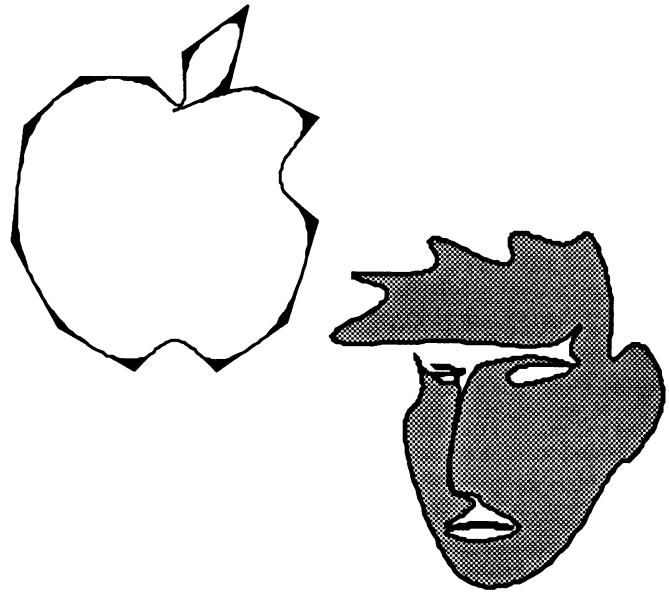


Figure 1. Sample Smoothed Picutres

A Different Approach

Before we get involved with smoothing, there is something to point out on sending *picComs* to the printing grafPort that I think is fundamental and seems to have been left out of any discussion. You probably noticed that all the examples published are structured something like this:

```
myPic := OpenPicture(theWorld);
  DrawStuff;          (with QD + picComments)
ClosePicture;
PrintThePicture;    (go print myPic)
```

This will work nicely in an example test or in a small program, but in a real application will probably give you some problem. Why? Very simple. Since you usually give the freedom to your users to draw as many polygons or rotated texts as would fit in the available heap, it can easily happen that there isn't enough memory to make a picture containing all graphics to be printed or saved.

So what to do? Can we divide the picture into smaller ones? If so we must create and kill pictures in the middle of the printing loop, and this is not allowed as specified in IM2 (p. 160) where they warn us: "Don't call the QuickDraw function *OpenPicture* [...] after a call to *prOpenPage* ...". Do we have to open a picture in order to use *picComs*? Although we usually must, when printing we do not really have to.

Let's have a look at some facts now and see why this is

possible. When we are spool printing on an imagewriter, the driver is actually writing a PICT format file on the disk (or possibly to memory). To accomplish this, it has to solve the same problem we just talked about, as nothing guarantees that it will have enough memory to accumulate the picture and eventually print it. Fortunately, it is possible to customize the standard *QDprocs*, the low-level bottleneck drawing routines of QD. The imagewriter driver customizes the *stdPutPic* procedure (in the *PrOpenDoc* call) in order to directly write the picture to disk while it is forming, instead of accumulating it in memory, and opens a picture in the *PrOpenPage* call. Since another picture is already opened you can't open one after, but just because of that we can issue our *picComs* without taking the trouble (and the memory) of making one by ourselves. The standard *commentProc* for the print port of the imagewriter will just ignore comments anyway.

I know you are wondering what if we are printing on a laser, since it usually prints in a *draft* mode. Ok, we don't have a picture opened and we are not writing a PICT to the disk, but the driver has customized the *commentProc* that, as you guessed, is in charge of processing *picComs* causing them to be directly intercepted and sent to the laser without being accumulated anywhere.

All this simply means that you can safely issue *picComs* in your *page loop* without having to pass through the construction of a picture, this allowing you to print everything you want without worrying about memory problems besides making your print job faster and easier.

So our previous routine should look something like this:

```
PrOpenPage(MyPrintPort,NIL);        (don't need to open a pic-
ture)
  DrawStuff;  (using QD + PicComments)
PrClosePage(MyPrintPort);  (end cur page)
```

Smooth Commenting

When we have to smooth polygons we usually write a routine that draws many short line segments shaping the curve. We can use it to draw on the screen or in the printing *grafPort* of the imagewriter, but no matter how good our smoothing algorithm is, when we attempt to send it to a laser printer the results are far from being attractive. This is because the smooth-poly will always be printed at a 72 dpi resolution, making your hard copy much less impressive than it could be. This, of course, is before using *picComs* for your poly (sorry if it sounds like an ad).

Using *picComs* is not only useful for printing but very recommendable for saving (in a file or in the *scrap)* a *PICT* containing objects that can be better explained adding comments. Doing it in this way will allow other applications to reconstruct the objects in the right format. To simplify this, imagine that you have just saved in a picture file the short line segments forming the smooth-poly. An application like Mac Draw does not know that you do not want a very fractal object or just a number of connected lines, thus treating your masterpiece like a senseless doodle. Using *picComs* you can tell MacDraw that those short lines belong to a smoothed polygon.

Although the whole mechanism is pretty easy (once you know it) there are some rules to follow. The final goal of our discussion is to write a routine that is not concerned with the printer type but that takes advantage of specific situations without risk of being device dependent. This routine could also be called in order to save (on disk) or to copy a picture (in the *scrap*). In fact it is indifferent to send our *picComs* and drawings to the printer, to the disk or to the memory.

It is important to notice that some applications, like MacDraw (and our demo), do not always use a standard QD polygon (a 10 bytes header plus 4 bytes for each point), but a compressed version. When several polygons with many sides are in memory, the use of compressed polys could save a lot of memory. If, for example, you use a byte offset format nearly half the space can be saved. It is not convenient then to convert this object to a standard Poly in order to draw it using *stdPoly*; in fact it will be more trouble and take more code and memory than if you simply drew it using *stdLines*. Subsequent calls to *Lineto* or *Line*, after the *PolyBegin* comment, will be interpreted as part of the poly.

Let's now have a closer look at this routine. We always start the poly definition with a *PolyBegin* comment. This informs a driver, prepared to understand *picComs*, that the following *stdLines* (or *stdPoly*) are to be considered as part of a special polygon. Immediately following this comment we might issue the *PicPlyClo* comment which specifies that the polygon is a closed figure.

This header is the same for smoothed or unsmoothed polys, but at this point we should differentiate our code. If the current poly is not to be smoothed we can just draw it normally which would work for every printer; but if it does have to be smoothed we should issue additional information which is done through another *picCom*, the *polySmooth*. This comment takes a handle to one byte of data specifying the nature of the splined poly: the *verbs* parameters. These *verbs* provide a way to apply *framing*, *filling* and *closing* to the current object.

Right after this last comment we should draw the unsmoothed poly. Notice that we draw the original poly even though we want the smoothed one, since the laser only cares about the original vertexes on which will be applied its own smoothing algorithm (a cubic spline, if you care). If you are wondering, what if an imagewriter gets these vertexes instead of the smoothed ones, you have probably guessed that it would print the wrong thing, so we have to find a way not to have it print on paper. The official Apple documentation suggests setting the pen size to zero after the *polyBegin* comment. Unfortunately this method doesn't work properly. It does avoid printing the unsmoothed poly on a printer that is not equipped to handle polygon comments but, too bad, even on the laser the pen remains at zero and nothing will print.

The real solution has to do with clipping. What we have to do is simply set the clipping to a zero rect (a rectangle enclosing no pixels) before drawing the unsmoothed poly and then reset it to the previous clipping when finished. This will prevent an imagewriter from drawing the wrong thing, but will allow a laser, that ignores the clipping rectangle (when in a special poly definition), to draw the smoothed polygon.

• It is interesting to note that a picture with the clipRgn set to

an empty rect does not generate any opcode for this operation since such a clipping is actually its default •

At this point we are done with the laser, but nothing would have been printed yet on an imagewriter, so we must get ready to call our own smoothing algorithm. However, first we must ensure that the laser won't draw again. This is very easily done by the *polyIgnore* comment that simply puts the laser driver to sleep (actually only drowsing) until a *polyEnd* is encountered. Is that all? Basically this is it, but there are a few other things we should examine to completely master the whole scheme.

If the polygon has to be filled, as well as using the *fill verb* in the *PolySmooth* comment, the QD procedure *fillRgn* can be used. This comes after the *PolyIgnore* and will work for every printer. In fact the laser driver, even though it ignores regions, will be awakened from its sleep and will build a *fillPat* for its current *path* using that passed to the *FillRgn* proc. Some of the patterns have been optimized in order to translate them in a gray-scale that PS can halftone. The actual patterns that the driver transforms in a gray-scale, are the standard fill tones present in the system file (and as QD globals) plus some of the patterns in the standard list of the system file: to be exact those with numbers 1 to 5 (2=5) and 20 to 24 (23=24). All these patterns can be found as resources (PAT# ,ID=-8191) in the laser driver where, two more pats in the list (8 and 18), raise the total available shades of gray to the number of 11 including black and white. You can, therefore, easily put up a pattern menu with more real PS gray scales than most applications without having to deal with special coding or, in the worst case, having to write a driver yourself. In addition these patterns are much faster than others because they don't need to be imaged before being sent to the laser. Using these pats you can have gray-scales from white to black at approximately 10% increments in gray percentage. Currently there is a gap between 50 and 75 percent and there are 3 gray levels between 75% and 82% all very similar, so let me put on the wish list a gray around 60% instead of one of these last 3.

Let's summarize all this with a short pseudo-language example assuming our polygon is stored in a byte offset format, therefore using *Line* or *LineTo* procedures in its drawing routines. Shouldn't be hard to adapt this example in order to use normal QD polys.

```
PrOpenPage(MyPrintPort,NIL);  {check err}
  {Begin special poly}
  PicComment(PolyBegin,0,NIL);
    PicComment(picPlyClo,0,NIL);
    IF smoothed THEN BEGIN  {smoothPoly}
      {verb is an unsigned byte}
      IF noFill THEN SmoothHdl^^.verb=5;
      {close+frame+fill (4+1+2)}
      ELSE SmoothHdl^^.verb=7;
      {set smooth flag and verb}
      PicComment(Polysmooth,1,SmoothHdl);
      GetClip(saveClip);
      {laser don't care of Clip}
      ClipRect(ZeroRect); {but others do}
      {feed laser with original vertexes}
      DrawUnsmoothedPoly;
      SetClip(saveClip);  {restore}
      {laser can take a nap now}
      PicComment(PolyIgnore,0,NIL);
    END;  {smoothPoly}
                IF  filled  THEN  BEGIN
{make a rgn to fill our special poly}
                          OpenRgn;   {assuming  we  have  a
handle}
                                     DrawCurPoly;   {create re-
gion}
      CloseRgn(RgnHandle);
      FillRgn(RgnHandle,itsPat);
    END;
    DrawCurPoly;  {redraw}
  PicComment(PolyEnd,0,NIL);
  {End special poly}
PrClosePage(MyPrintPort); {end cur page}
.....................
PROCEDURE DrawCurPoly;
  {decide what routine to call}    IF smoothed THEN
DrawSmoothedPoly;
  ELSE DrawUnsmoothedPoly;
END;
```

Les jeux sont fait; this routine will print on any kind of printer regardless of whether it knows how to interpret *picComs* or if the current poly is smoothed or unsmoothed. The scheme represented in this routine is similar to that used by MacDraw and this guarantees that it will be supported for a long time.

As we said, besides to print, we can use this routine to store in the *scrap* or to save a PICT file on the disk and we can even use it to draw on the screen (although it's slower than normally). When we want to save a picture in the *scrap* we open a picture and then call our routine causing drawings and *picComs* to be accumulated. In order to inform an application receiving our PICT that it contains comments, we should bracket the special poly in two additional comments: *picDwgBegin* and *picDwgEnd*. Although these comments were designed for MacDraw, some other application could count on them, so even if they might be useless we should include them.

Let's now see a slightly different way to accomplish the same task which is derived from the new scheme used in MacDraw II. Basically it doesn't differ much from its predecessor except for the method used to fill the poly. As we recall from the previous example a region was build to fill the *pseudo-poly* whether it was smoothed or not, but, as we'll see now, a standard QD polygon can be used for the same purpose.

Using a QD poly, of course, involves building one. Keeping track of its size is a much more easier thing to do than for regions. In fact if you inconsiderately trespass the 32K limit of regions and polygons some really undesirable things will happen soon. How too make sure this won't happen using a region is (too bad) beyond the scope of this article but, as you probably guessed, checking for this kind of error while a polygon is forming is a much easier task: just make sure you don't have more then 8189 points or, using *GetHandleSize*, check for the 32K limit not to be passed. The original MacDraw ignores a fill request for a polygon that does not fit in a 32K region. This means that if you try to fill a big and complicated smoothed poly MacDraw refuses to do it. Surprisingly MacDraw II doesn't take advantage of a much easier size checking derived by using polygons and dies in a horrible way when overloaded.

Here is how it could look like our previous routine modified

in order to use the procedure *FillPoly* instead of *FillRgn*:

```
PrOpenPage(MyPrintPort,NIL); {check err}
  PicComment(PolyBegin,0,NIL);
  IF Fill THEN BEGIN  {build Poly}
    ThePoly := OpenPoly;
      DrawCurPoly; {accumulate lines}
    ClosePoly;
  END;  {build Poly}
    IF smoothed THEN BEGIN  {smoothPoly}
      IF noFill THEN SmoothHdl^^.verb=5;
      ELSE SmoothHdl^^.verb=7;
      PicComment(Polysmooth,1,SmoothHdl);
    END;  {smoothPoly}
    IF Fill THEN FillPoly(ThePoly,ThePat);
    GetClip(saveClip);
    ClipRect(ZeroRect); {laser ignores this}
    DrawUnsmoothedPoly; {original vertex}
    SetClip(saveClip);  {restore}
    PicComment(PolyIgnore,0,NIL);
    DrawCurPoly;  {draw poly frame}
  PicComment(PolyEnd,0,NIL);
PrClosePage(MyPrintPort); {end cur page}
```

Notice that, as shown in the other example, the procedure DrawCurPoly checks whether the poly is smoothed and, if this is so, calls the smoothing algorithm, otherwise draws the lines shaping the poly using original vertexes. When we call DrawCurPoly right after the PolyIgnore *picCom* it might also be possible to issue a *FramePoly* QD *proc* instead. In this case nothing guarantees you were able to build one since it might have happened that the poly exceeded the size limit making its creation impossible. Using *stdLines* always guarantees you will be able to frame your object.

MacDraw II actually does a few other things than in this example. Those things seem completely useless to me, so I took them out of the example test that demonstrated to work well even without those cryptic statements. In fact MacDraw II would start moving the pen out of the way (-8000,-8000) and then setting the pensize to zero, however, after the *FillPoly proc* (before any intervening drawing), it would reset the pensize and location cancelling the previous operation. Mac Draw II uses a zero region to fill the poly (as we discussed before). The laser driver is not concerned with the region data but uses only the fill pattern passed along with the procedure. This, again, is useless since the driver would have already taken the pattern at the issue of the *FillPoly proc* in order to fill its current path. Note that MacDraw II passes a 2 bytes verb in the PolySmooth *picCom* instead of the only one used originally. However this second byte seems not to be used and cleared to zero at the moment.

Although there might be some invisible reason (at least to me) for MacDraw II to use this additional statements, you can safely use the slimmed version we just saw or the scheme used by the original MacDraw shown before (and also presented in the demo). Both of These methods have demonstrated to work fluently in all the tests I produced. They also worked fine in order to save a Pict file format on disk. In this case we must write a file having cleared to zero its first 512 bytes block (usually containing MacDraw specific information) followed by our picture. As we said for the imagewriter driver, we may not have enough memory to accumulate the picture in memory before writing it to disk, so we should customize the *stdPutPic* in order to directly spool it to disk while it is forming.

Real Width

Let's now have a look to another interesting *picCom* which has the ability to scale the pen size even under the smallest unit (1/72 of an inch) that QD can represent. We are talking about the *SetLineWidth* (from now on *SLW*) comment that allows you to draw with an almost infinitely thin pen, taking advantage of higher resolution laser printers.

This is a very important comment that, I think, every application dealing with graphics should use in order to give its users the possibility to fully utilize laser capabilities.

SLW takes a handle to a 4 bytes data point which is actually to be interpreted as a real number. The first high order integer (2 bytes) is the numerator and the second the denominator of a fraction giving the scaling factor for the current pen. For example if you pass 1 for the numerator and 4 for the denominator you will have a value of 0.25 (1/4) that will be used to determine the current pen for the *SetLineWidth* PS operator.

• Don't confuse the *SetLineWidth picCom* with the homonymous PS operator which actually assigns the pensize for the laser •

The LaserPrep file contains variables and procedures that the driver downloads to the printer. The *SLW picCom* works by changing the value of some of these variables. There is a global variable flag string (*fg*) containing single-character values; the fourth and the fifth parameters of this string represent respectively the numerator and the denominator of a variable named *pnm*. The *SLW picCom* stores the two parameters it receives in these character bytes therefore limiting to an unsigned byte (0-255) the maximum possible value that can be passed. Note that you pass numbers in integer format but they are interpreted as unsigned bytes. If for example you wish to scale your pen to 300 dpi (instead of the 72 default), it would be fatal to issue a *SLW* having 72 in the first word and 300 in the second ($ 0048 012C). In fact, the number 300 will not fit in the unsigned byte denominator of *pnm* which would cause the driver to print nothing or to bomb totally (depending on version). To perform such an operation, scale the number to a common factor like 12 for example, giving a fraction of 6/25 which will scale the pen exactly like 72/300 would have done.

Unfortunately the *SLW picCom* is very tricky and often doesn't appear to work as expected until you know its idiosyncrasies. Let's see the inner workings of this comment in conjunction with the laser driver.

• For simplicity we'll consider width and height of QD pen as a unique parameter •

Any time you issue a *SLW* picCom, in the printing loop, the laser driver translates it into a call to the previously downloaded *lw* (line width) procedure. The *lw* proc calculates the width of the pen that is used for the *SetLineWidth* PS operator. If we name 'LineWidth' the variable containing the thickness of lines to be drawn (what an imagination) and 'pnmV' and 'pnmH' are the two parameters you pass in the *SLW picCom*, the *lw* procedure works about like this: LineWidth = LineWidth * (pnmV/pnmH).

Then the *SetLineWidth* PostScript operator will take LineWidth as its argument setting the pen for the laser to this new value. This means that every new scaling factor you pass through the *SLW* comment will not actually act on the current QD pensize, but will multiply the PS pen. For example if you have a starting QD pensize of 2 and you pass a *pnm* of 3/2 (1.5) the PS pen will be set to 3 and this is no surprise, but if a further *SLW* is generated with a *pnm* of 1/2 (0.5) we will have a new pen of 1.5 (3 * 0.5) and not 1 as would have resulted multiplying the QD pen (2) with the new scaling (0.5). So one more *SLW* comment with a *pnm*=0.5 will set the PS pen to 0.75 leaving unchanged the QD pen (2).

Further more we must not forget that even the *pensize* QD procedure affects the laser pen width, but differs from the *SLW picCom*, in that it is not directly processed and sent to the laser. In fact until a new drawing routine is called, arguments passed to *pensize* are only recorded but no action is taken. When a QD bottleneck is called the driver checks the last value passed to pensize and, if it's different from the previously used value, it generates a '*pen*' LaserPrep's routine. Imagining the new QD *pensize* to be a unique value (for width and height) named NewSize this procedure would look about like this: LineWidth = NewSize * (pnmV/pnmH). LineWidth will be then passed to the *SetLineWidth* PS operator.

This procedure will have the effect of resetting the laser pen according to the new QD pen and the current *pnm*. So, considering we left our PS pen to 0.75 and the *pnm* to 0.5 as in the previous example, making a call to *PenSize* with an argument of 1 would cause the pen (when a QD proc is called) to be set at 0.5, indeed NewSize * pnm (1 * 0.5).

Notice that, because the laserPrep's '*pen*' procedure is called right before a QD routine (if at all), it is always issued after the '*lw*' procedure even if the original order was the opposite.

So many things just to set a pen width? Don't ask me, but once you figure out the mechanism it shouldn't be impossible to make it work. I suggest that, following the rules we have just talked about, you write a short routine in your favorite language to test all the quirks of this comment.

Here's a pseudo one that can serve as a skeleton and assumes (for simplicity) the QD pen to be a unique value:

```
LineWidth  = Float  (PS pen)
QDPenSize  = Integer  (QD cur pen size)
pnm = Float  (pen multiplier)
NewSize  = Integer  (last val of PenSize)
.....................
(Everytime a SLW picCom is issued call the followin proc to
simulate the driver)
PROCEDURE Lw (pnmH,pnmV);
  pnm := pnmV / pnmH;
  LineWidth := LineWidth * pnm;
END; (Lw)
.....................
(If a QD PenSize is called just set NewSize accordingly)
(Before drawing, check IF QDPenSize <> NewSize THEN (in case)
call this routine)
PROCEDURE Pen;
   QDPenSize := NewSize;
  LineWidth := QDPenSize * pnm;
END; (Pen)
.....................
(You can then print LineWidth, instead
drawing, to test the PS pen size )
```

About the Demo

The principle goal of our Demo is to show how to correctly insert in a working program the routines we have been talking about.

The program allows a polygon to be drawn (in a non modal way) in a standard Macintosh window, you can then take some action on it through menus. It is possible to smooth and vice-versa, fill the poly with some patterns and change the pen size. Moreover you can copy the current poly in the clipboard and, because desk accessories are supported, you can then paste it in to those of them accepting picts or in other applications. Of course you can print the current poly on any printer you wish and setting the pen scaling factor for PostScript printers will allow a much more accurate definition for the pen width like drawing hairlines as thin as possible on the currently attached device. For example a pen size of 3 with a scaling factor of 300 dpi will cause lines to be printed at a width of 1/100 of an inch (3 * 1/300) which on a laser writer's raster memory will be exactly 3 pixels.

Although the program only serves as a demo its routines have been designed to be fast and compressed in order to be perfectly usable in a larger project. For example the actual procedure in charge of smoothing an input poly can be considered like a Toolbox routine for its speed and compression. You could even be disinterested in how this routine works (of which a full discussion would go beyond the scope of this article), and you could just call it from your program passing the right arguments like you would do with any Toolbox proc. Seemingly the routine that builds a special polygon ('MakePoly') can be easily used from a different application even if dealing with hundreds of smoothed polys instead of the only one we can have in the demo. Points where potential problems may arise have been marked and a short description of the possible bug is given. This, of course, is done not for carelessness but because a lot of code would have been needed and we couldn't take up any more space. Assembly language has been chosen mainly for its insuperable speed since we all experienced what it means in terms of time to scroll a certain number of smoothed polygons in a window. Furthermore having to deal with compressed data structures makes it much easier to control them in this language. *Macros* have been used for comments dividing them in short and long as in the picture's opcodes. Routines that refer more closely to the subject of this article, as you have seen, have already been shown in a pseudo high-level language. Hierarchical menus have been used where appropriate and the new *PrGlue* Toolbox trap (in 256K ROMs and in system 4.1 or later) is used for printing code.

I hope this discussion saved you some time in implementing comments in your applications. Please note that despite most of the information being presented in this article is not documented anywhere else, it is not a weird approach to accomplish the work done but just the way things are at the moment and probably will be for a while.

Besides the following documentation MacsBug was of

invaluable help as usual.

Bibliography

Inside Macintosh™, Addison-Wesley Publishing Company Inc. 1985-88, Vol.I, chap. 6, p.159, 189; Vol.II, chap. 5; Vol.V chap. 4, pp.85-105

APDA, *Apple Laser Writer Reference Manual*, Apple Computer Inc., Cupertino CA 1987

APDA, *Macintosh Technical Notes 1988*, notes: #21, #27, #72, #91, #122, #175, #181

Adobe Systems Inc., *PostScript Language Reference Manual*, Addison-Wesley Publishing Company Inc. 1985, 7th Printing 1987

Adobe Systems Inc., *PostScript Language Tutorial and Cookbook*, Addison-Wesley Publishing Company Inc. 1985, 7th Printing 1987

Adobe Systems Inc., *PostScript Language Program Design*, Addison-Wesley Publishing Company Inc. 1988

Dan Weston, *The Complete Book of Macintosh Assembly Language Programming*, Scott, Foresman & C., Glenview, Illinois London 1987, Vol.II, chap. 4.

Scott Knaster, *Macintosh Programming Secrets*, Addisoison-Wesley Publishing Company Inc. 1988, 2nd Printing, chap. 6

Dave Kelly, David E. Smith, *Pascal Procedures. MultiFinder Friendly MacDraw Plotter*, «MacTutor», vol.4, n.2, February 1988, pp.16-41

Joel West, *Comments About Picts*, «MacTutor», vol.4, n.6, June 1988, pp.40-48

Listing: Smooth.Link

```
/Output Smoother

/Type 'APPL' 'DEMO'
/Bundle
Smooth

/Resources
SmoothRes

/End
```

Listing: Smooth.asm

```
; ---------------------------
; - Smoother: picCommenting polygons & non solo ------
; - Raul Tabasso - via delle Isole 23/A - 00198 ROMA --
; ---------------------------
; WARNING: because this program is intended as a demo, there
;  are some circumstances where a stronger error checking
;  should be applied. Due to this, situations where a
; potential problem may arice,
;   are marked with a (¿) symbol and a short advice is given.
; ---------------------------
Include  Inside comments:SmoothSource:Traps.D
; ---------------------------
MBarHeight      EQU     $BAA            ;GENERAL EQUATES
DoubleTime      EQU     $2F0
screenBits      EQU     $FF86
PortRect        EQU     16
; ---------------------------
PrGlue          EQU     $A8FD           ;New print manager call
PrOpen          EQU     $C8000000       ;PrGlue's routine selec-
tors
PrClose         EQU     $D0000000
PrintDefault    EQU     $20040480
PrStlDialog     EQU     $2A040484
PrJobDialog     EQU     $32040488
PrOpenDoc       EQU     $04000C00
PrCloseDoc      EQU     $08000484
PrOpenPage      EQU     $10000808
PrClosePage     EQU     $1800040C
PrPicFile       EQU     $60051480
PrError         EQU     $BA000000
; ---------------------------
iCopies         EQU     4               ;Print Manager Stuff
prInfo          EQU     2
prJob           EQU     62
rPage           EQU     6
bJDocLoop       EQU     6
; ---------------------------
PicDwgBeg       EQU     130             ;PicComment Kinds
PicDwgEnd       EQU     131
PolyBegin       EQU     160
PolyEnd         EQU     161
PolyIgnore      EQU     163
PolySmooth      EQU     164
PicPlyClo       EQU     165
SetLineWidth    EQU     182
; ---------------------------
Frame           EQU     1               ;PolyVerbs
Fill            EQU     2
Close           EQU     4
; ====================================
; ------ M A C R O ------
; ---------------------------
MACRO   prExec RoutineSelect=
  MOVE.L #(RoutineSelect),-(SP)     ;routine selector on stack
  DC.W   PrGlue                 ;execute trap
  |
; ---------------------------
MACRO   ShortComment Kind   =
  MOVE.W #(Kind),-(SP)
  CLR.W  -(SP)
  CLR.L  -(SP)
  _PicComment
  |
; ---------------------------
MACRO   LongComment  Kind,Size,Hand      =
  MOVE.W #(Kind),-(SP)
  MOVE.W #(Size),-(SP)
  MOVE.L (Hand),-(SP)
  _PicComment
  |
; ---------------------------
MACRO   CheckItm     MHandle,Item =
  MOVE.L (MHandle)(A5),-(SP)
  MOVE.W (Item),-(SP)
  ST     -(SP)                  ;check
  _CheckItem
  |
; ---------------------------
MACRO   UnChkItm     MHandle,Item =
  MOVE.L (MHandle)(A5),-(SP)
  MOVE.W (Item),-(SP)
  CLR    -(SP)                  ;uncheck
  _CheckItem
  |
; ====================================
; ---- Register usage ------
PolyHand        EQU     A4
; ---------------------------
  BSR   InitGlobals             ;Do not rearrange the
  BSR   InitMac                         ;order of these subs
  BSR   GetHandles
  BSR   SetMenus
  BSR   SetWindow
  BRA   NewPoly                         ;init one
```


© The Definitive MacTutor, Vol. 4

```
; ====================================
Loop     _SystemTask
  CLR.W  -(SP)
  MOVE.W #$FFFF,-(SP)        ;Every event
  PEA    EVENTRECORD(A5)
  _GetNextEvent
  MOVE.B (SP)+,D0
  BNE.S  DoEvent
  TST.W  whileDrawing(A5)          ;Check if we have to…
  BNE.S  TrackLine           ;Idle line
  BRA.S  Loop

DoEvent  MOVE.W WHAT(A5),D0
  CMP.W  #1,D0               ;is mousedown ?
  BEQ    MouseDown
  CMP.W  #3,D0               ;is keydown ?
  BEQ    KeyDown
  CMP.W  #6,D0               ;is update ?
  BEQ    UpDate
  CMP.W  #8,D0               ;is activate ?
  BEQ    DoAct
  BRA.S  Loop                ;we don't support other events
; ========================================
TrackLine
  PEA    CurPoint(A5)
  _GetMouse
  MOVE.L CurPoint(A5),D0
  CMP.L  OldPoint(A5),D0     ;Did we move since last time ?
  BEQ.S  Loop
                             ;we are in XOR mode
  MOVE.L LastPoint(A5),-(SP) ;start Pt
  _MoveTo
  MOVE.L OldPoint(A5),-(SP)  ;overwrite old line
  _LineTo
  MOVE.L LastPoint(A5),-(SP) ;draw new line
  _MoveTo
  MOVE.L CurPoint(A5),-(SP)
  _LineTo
  MOVE.L CurPoint(A5),OldPoint(A5) ;c'est la vie
  BRA.S  Loop
; ------------------------
DoAct    MOVE.W MODIFY(A5),D0
  BTST   #0,D0               ;Activate or deactivate
  BEQ.S  DeAct
  MOVE.L MESSAGE(A5),-(SP)
  _SetPort
  BSR    DimMenu             ;we only support copy menu
  BSR    PutGrow
  BRA    LOOP
DeAct    BSR    ActMenu      ;Allow complete edit menu for DA
  PEA    GrowBox(A5)
  _EraseRect                 ;clean it and
  BSR    PutGrow             ;draw inactive
  BRA    LOOP
; ------------------------
UpDate
  MOVE.L TheWind(A5),-(SP)
  _BeginUpdate
  TST.W  PolyDrawn(A5)
  BEQ.S  @0                  ;branch if no poly
  BSR    ReDraw              ;show poly
  BRA.S  @1
@0TST.W  WhileDrawing(A5)
  BEQ.S  @1            ;Exit if we are not either drawing
  BSR    DrawPoly            ;Redraw part already drawn
  MOVE.L LastPoint(A5),-(SP) ;redraw idle line   _MoveTo
  MOVE.L OldPoint(A5),-(SP)
  _LineTo
@1BSR    PutGrow
  MOVE.L TheWind(A5),-(SP)
  _EndUpdate
  BRA    LOOP
; ------------------------
MouseDown
  CLR.W  -(SP)
  MOVE.L WHERE(A5),-(SP)
  PEA    WINDOW(A5)
  _FindWindow
  MOVE.W (SP)+,D0
  MOVE.W D0,D3               ;needed for zoom box
  ADD.W  D0,D0
  MOVE.W WTABLE(PC,D0.W),D0
  JMP    WTABLE(PC,D0.W)
WTABLE   DC.W   LOOP-WTABLE
  DC.W   DoMenu-WTABLE
  DC.W   SysEvnt-WTABLE
  DC.W   DoContent-WTABLE
  DC.W   DoDrag-WTABLE
  DC.W   DoGrow-WTABLE
  DC.W   Loop-WTABLE
  DC.W   DoZoom-WTABLE
  DC.W   DoZoom-WTABLE
; ------------------------
KeyDown  CLR.L  -(SP)
  MOVE.W MESSAGE+2(A5),-(SP)
  _MenuKey
  MOVE.L (SP)+,MENU(A5)
  BRA.S  ExMenu
; ------------------------
DoMenu
  CLR.L  -(SP)
  MOVE.L WHERE(A5),-(SP)
  _MenuSelect
  MOVE.L (SP)+,MENU(A5)
ExMenu   CLR.W  -(SP)
  _HiliteMenu                ;Hilite off
  MOVE.W MENU(A5),D0
  CMP.W  #4,D0               ;is a submenu ?
  BLE.S  @0
  SUB.W  #'A'-4-1,D0   ;Allign Submenu item-60 (65-4-1)
@0ADD.W  D0,D0
  MOVE.W MTABLE(PC,D0.W),D0
  JMP    MTABLE(PC,D0.W)
MTABLE   DC.W   LOOP-MTABLE          ;Menus
  DC.W   DoMela-MTABLE
  DC.W   DoFile-MTABLE
  DC.W   DoEdit-MTABLE
  DC.W   DoDo-MTABLE
  DC.W   DoPensize-MTABLE            ;submenus
  DC.W   DoPat-MTABLE
  DC.W   DoPenscale-MTABLE
; ------------------------
DoContent
  CLR.L  -(SP)
  _FrontWindow
  MOVE.L (SP)+,D0
  CMP.L  WINDOW(A5),D0
  BEQ.S  @0                  ;Already in front
  MOVE.L WINDOW(A5),-(SP)
  _SelectWindow
@0BSR    ChkDoubClick        ;set DoubleClick(A5) accordingly
  MOVE.L WHERE(A5),MousLoc(A5)
  PEA    MousLoc(A5)
  _GlobalToLocal

  TST.W  PolyDrawn(A5)       ;do we have a poly yet ?
  BNE    Loop                ;yes, no action implemented
  TST.W  whileDrawing(A5)    ;No, is it initialized yet ?
  BNE    AddPoint            ;yes, one more line for the poly
  BRA    PolyInit            ;No, start a new one
; ------------------------
DoDrag   MOVE.L WINDOW(A5),-(SP)
  MOVE.L WHERE(A5),-(SP)
  PEA    WBounds(A5)
  _DragWindow
  BRA    LOOP
; ------------------------
DoGrow   CLR.L  -(SP)
```

```
  MOVE.L WINDOW(A5),-(SP)
  MOVE.L WHERE(A5),-(SP)
  PEA    WSize(A5)
  _GrowWindow
  MOVE.L (SP)+,D0
  BEQ    LOOP                ;Laizy, if not grown

  MOVE.L WINDOW(A5),-(SP)
  MOVE.L D0,-(SP)
  ST     -(SP)
  _SizeWindow
  PEA    GrowBox(A5)
  _EraseRect                 ;Erase it and
  PEA    GrowBox(A5)         ;put it in the update rgn
  _InvalRect

ResetWnd BSR   CalcGrow
  BSR    ClipWind
  BSR    PutGrow
  BRA    LOOP
; ---------------------
DoZoom   CLR.W  -(SP)
  MOVE.L WINDOW(A5),-(SP)
  MOVE.L WHERE(A5),-(SP)
  MOVE.W D3,-(SP)            ;PartCode in or out
  _TrackBox
  TST.W  (SP)+
  BEQ    LOOP
  MOVE.L WINDOW(A5),A0
  PEA    PortRect(A0)
  _EraseRect
  CLR.W  -(SP)
  MOVE.L WINDOW(A5),-(SP)
  MOVE.W D3,-(SP)            ;PartCode in or out
  SF     -(SP)
  _ZoomWindow
  BRA    ResetWnd
; ---------------------
SysEvnt  PEA   EVENTRECORD(A5);make system happy
  MOVE.L WINDOW(A5),-(SP)
  _SystemClick
  BRA    LOOP
; ---------------------
PolyInit
  MOVE.L MousLoc(A5),-(SP)   ;Start poly
  _Moveto
  MOVE.W #10,-(SP)           ;XOR mode while drawing poly
  _PenMode
  MOVE.L #$10001,-(SP)
  _PenSize                        ;Pen 1,1
  MOVE.W #1,NumPoints(A5)         ;First point
  MOVE.L (PolyHand),A0
  MOVE.L MousLoc(A5),(A0)         ;write it in the list
  MOVE.W #4,BytesWritten(A5)
  ST     whileDrawing(A5);follow mouse movement with a line
  MOVE.L MousLoc(A5),LastPoint(A5)  ;init points
  MOVE.L MousLoc(A5),OldPoint(A5)
  BRA    Loop
; ---------------------
AddPoint
  TST.W  DoubleClick(A5)     ;user wants to close the poly ?
  BNE    ClosePoly
  MOVE.L LastPoint(A5),-(SP) ;clear last line
  _MoveTo
  MOVE.L OldPoint(A5),-(SP)
  _LineTo
  _PenNormal                 ;Reset pen
  MOVE.L LastPoint(A5),-(SP) ;add new line
  _MoveTo
  MOVE.L MousLoc(A5),-(SP)
  _LineTo
  MOVE.W #10,-(SP)           ;put back XOR mode
  _PenMode
  BSR    DoAdd               ;put new point into the list
  CMP.W  #32760,BytesWritten(A5)   ;Do not write past block
  BPL    ClosePoly           ;force to close in case
  BRA    Loop
; ---------------------
; Pack 'MousLoc' into compressed poly data list :
; A negative word signals a short delta; a word as follows…
; / bit 15 always set / bits 14-8 = dy (-64,63) / bits 7-0 =
dx
; (-128,127)/ A positive word signals a 4 bytes point (Y & X)
; as usual WARNING(¿): this scheme won't work if high bit of
; point is negative(if a Y becomes <0 a point will be
; interpreted as a short offset)
; ---------------------
DoAdd    ADDQ.W #1,NumPoints(A5)          ;one more point
  MOVE.L (PolyHand),A0       ;Pointer in A0
  ADDA.W BytesWritten(A5),A0 ;get to mark

  MOVE.W MousLoc(A5),D0             ;current Y
  SUB.W  LastPoint(A5),D0           ;D0 = Delta Y
  CMP.W  #63,D0              ;Dy must be in the range
  BGT.S  LongPoint           ;doesn't fit
  CMP.W  #-64,D0
  BMI.S  LongPoint
                             ;Y fits, now check X
  MOVE.W MousLoc+2(A5),D1    ;current X
  SUB.W  LastPoint+2(A5),D1  ;D1 = Delta X
  CMP.W  #127,D1             ;Dy in a signed byte range ?
  BGT.S  LongPoint
  CMP.W  #-128,D1
  BMI.S  LongPoint
WordPoint                    ;Word format Point (compressed)
  ADD.W  #2,BytesWritten(A5) ;only 2 bytes for a short offset
  BSET   #7,D0               ;force a negative word to signal
  MOVE.B D0,(A0)+            ;write Y offset
  MOVE.B D1,(A0)                   ;write X offset
  BRA.S  AddDone
LongPoint                    ;Long format (normal)
  ADD.W  #4,BytesWritten(A5) ;not an offset, a Point
  MOVE.L MousLoc(A5),(A0)    ;always positive in our plane
AddDone
  MOVE.L MousLoc(A5),LastPoint(A5)  ;update
  MOVE.L MousLoc(A5),OldPoint(A5)
  RTS
; ---------
ClosePoly
  ST     PolyDrawn(A5)       ;Poly is now completed
  CLR.W  whileDrawing(A5)    ;we are through
  _PenNormal                 ;reset mode and size
  MOVE.L DoHand(A5),-(SP)
  MOVE.W #1,-(SP)
  _EnableItem                ;allow unsmooth/smooth
  MOVE.L FileHand(A5),-(SP)
  MOVE.W #3,-(SP)
  _EnableItem                ;and print

  MOVE.L (PolyHand),A0       ;Pointer in A0
  MOVE.L (A0),MousLoc(A5);save extra code simulating a click
  BSR    DoAdd               ;at last point (MousLoc) = first
  _OpenRgn     ;WARNING :(¿) should make sure our poly is not
  BSR    DrawPoly            ;too big to fit in a region.
  MOVE.L RgnHand(A5),-(SP)   ;if it doesn't fit, we are dead
  _CloseRgn
  BSR    SetPen
  BSR    CleanWind
  BSR    ReDraw              ;show poly
  BRA    Loop
; ============================
DoMela
  CMPI.W #2,MENU+2(A5)
  BGT    @1
                             ;Show about dialog
  CLR.L  -(SP)               ;space for result
  MOVE.W #256,-(SP)          ;ID
  CLR.L  -(SP)               ;NIL storage
```


© The Definitive MacTutor, Vol. 4

```
   MOVE.L #-1,-(SP)          ;in front
   _GetNewDialog
   MOVE.L (SP)+,A2
@0CLR.L  -(SP)
   PEA    ItmHit(A5)
   _ModalDialog
   CMP.W  #1,ITMHIT(A5)
   BNE.S  @0
   MOVE.L A2,-(SP)
   _DisposDialog
   BRA    LOOP

@1MOVE.L MelaHand(A5),-(SP) ;Get DA
   MOVE.W MENU+2(A5),-(SP)
   PEA    DskName(A5)
   _GetItem
   CLR.W  -(SP)
   PEA    DskName(A5)
   _OpenDeskAcc
   MOVE.W (SP)+,D0
   BRA    Loop
;-------------------
DoFile
   MOVE.W MENU+2(A5),D0      ;Item Num
   ADD.W  D0,D0
   MOVE.W M1TAB(PC,D0.W),D0
   JMP    M1TAB(PC,D0.W)
M1TAB    DC.W   LOOP-M1TAB
   DC.W   NewPoly-M1TAB
   DC.W   PageSetup-M1TAB
   DC.W   Print-M1TAB
   DC.W   LOOP-M1TAB         ;Laser Pen Menu
   DC.W   Quit-M1TAB
;-------------------
DoEdit
   CLR.L  -(SP)              ;Check if DA in front
   _FrontWindow
   MOVE.L (SP)+,A0
   CMP.L  TheWind(A5),A0
   BEQ    DoScrap            ;branch if our wind in front
   CLR.B  -(SP)              ;System Wind Frontmost
   MOVE.W MENU+2(A5),-(SP)
   SUBQ.W #1,(SP)            ;balance different numbering
   _SysEdit
   TST.B  (SP)+              ;get rid of bool
   BRA    LOOP
;-------------------
NewPoly
   CLR.W  PolyDrawn(A5)      ;clear poly
   CLR.W  WhileDrawing(A5)
   CLR.W  NumPoints(A5)
   CLR.W  Liscio(A5)         ;no poly, no smooth
   MOVE.L DoHand(A5),-(SP)         ;Menu Handle
   MOVE.W #1,-(SP)
   PEA    'Smooth'
   _SetItem                        ;reset
   MOVE.L DoHand(A5),-(SP)
   MOVE.W #1,-(SP)
   _DisableItem              ;no smooth menu and
   MOVE.L FileHand(A5),-(SP) ;no print active
   MOVE.W #3,-(SP)
   _DisableItem
   BSR    CleanWind
   BRA    Loop
;-------------------
PageSetup
   prExec PrOpen             ;open the driver
   BSR    CheckErr           ;driver ok
   CLR.W  -(SP)              ;Style Dialog
   MOVE.L PrintRec(A5),-(SP)
   prExec PrStlDialog
   MOVE.W (SP)+,D0           ;forget it
   prExec PrClose
   BRA    Loop
```

```
; ======================================
Print
   MOVE.L D3,-(SP)           ;save D3
   PEA    SavePort(A5)       ;Save current port
   _GetPort
   prExec PrOpen             ;open the driver
   BSR    CheckErr           ;driver opened successfully ?

   CLR.W  -(SP)              ;Job Dialog
   MOVE.L PrintRec(A5),-(SP)
   prExec PrJobDialog
   MOVE.W (SP)+,D0
   BEQ    PrintDon           ;canceled?

   BSR    OroCurs            ;put watch curs
   CLR.L  -(SP)              ;Open Print Port
   MOVE.L PrintRec(A5),-(SP)
   CLR.L  -(SP)
   CLR.L  -(SP)
   prExec PrOpenDoc
   MOVE.L (SP)+,PrintPort(A5)
   BSR    CheckErr           ;problems?

   MOVE.L PrintRec(A5),A0    ;Initialize New Graph Port
   MOVE.L (A0),A0
   PEA    prInfo+rpage(A0)
   _ClipRect

   MOVEQ  #1,D3              ;assume Spool
   MOVE.L PrintRec(A5),A0
   MOVE.L (A0),A0
   TST.B  prJob+bjDocLoop(A0) ;Check if Draft or Spool
   BNE.S  @1
   MOVE.W prJob+iCopies(A0),D3;Set Num copies if Draft
@1SUBQ.W #1,D3               ;for DBRA

PageLoop
   MOVE.L PrintPort(A5),-(SP) ;get ready to draw
   CLR.L  -(SP)
   prExec PrOpenPage         ;new page
   BSR    CheckErr           ;are we ok?

   MOVE.L CurPen(A5),-(SP)          ;Set size and…
   _PenSize
   MOVE.L CommentHand(A5),A1  ;scaling
   MOVE.L (A1),A0             ;deference handle
   MOVE.W #12,(A0)+           ;numerator = 72/6 and…
   MOVE.W CurScale(A5),(A0)   ;denominator in dpi/6LongComment
SetLineWidth,4,A1

   BSR    MakePoly           ;do it

   MOVE.L PrintPort(A5),-(SP)
   prExec PrClosePage        ;page done
   BSR    CheckErr

   DBRA   D3,PageLoop        ;next page

   MOVE.L PrintPort(A5),-(SP)
   prExec PrCloseDoc         ;document done
   BSR    CheckErr

   MOVE.L PrintRec(A5),A0
   MOVE.L (A0),A0
   TST.B  prJob+bjDocLoop(A0) ;Check if Draft or Spool
   BEQ.S  PrintDon           ;Branch if draft

   MOVE.L PrintRec(A5),-(SP)  ;spool to printer
   CLR.L  -(SP)
   CLR.L  -(SP)
   CLR.L  -(SP)
   PEA    PrStatus(A5)
   prExec PrPicFile          ;print spool file
   BSR    CheckErr
```

```
PrintDon prExec PrClose      ;close driver
  MOVE.L SavePort(A5),-(SP)
  _SetPort                   ;restore port
  MOVE.L (SP)+,D3            ;register and
  _InitCursor                ;arrow curs
  BRA    Loop
; ---------
CheckErr CLR.W -(SP) ;WARNING (¿)
  prExec PrError       ;if an error occurs we have no handler
  MOVE.W (SP)+,D0            ;Err code in D0
  BNE    Error               ;this just goes to the finder
  RTS
; ==============================
Quit     _ExitToShell              ;ciao
; ==============================
DoScrap
  TST.W  PolyDrawn(A5)
  BEQ    Loop                ;nothing to clip

  PEA    WRect(A5)           ;(¿) Not a correct rect
  _ClipRect                  ;it's here just to save code

  CLR.L  -(SP)
  PEA    WRect(A5)     ;(¿) should use a better rect
  _OpenPicture         ;(¿) Do we have enough mem for a pict?
  MOVE.L (SP)+,A2            ;PicHandle
  ShortComment PicDwgBeg     ;yes, it's the format you know
BSR      MakePoly            ;do it
  ShortComment PicDwgEnd
  _ClosePicture

  BSR    ClipWind            ;restore clip
  CLR.L  -(SP)
  _ZeroScrap
  MOVE.L (SP)+,D0
  BNE    Error
  MOVE.L A2,A0
  _GetHandleSize             ;how big is the pict
  CLR.L  -(SP)
  MOVE.L D0,-(SP)            ;lenght
  MOVE.L #'PICT',-(SP)       ;ResType
  MOVE.L (A2),-(SP)          ;Ptr Pict
  _PutScrap                  ;put it down
  MOVE.L (SP)+,D0
  BNE    Error
  MOVE.L A2,-(SP)
  _KillPicture               ;Get rid of it
  BRA    Loop
; ---------------
DoDo                               ;Smooth or unsmoooth
  MOVE.L DoHand(A5),-(SP)          ;Menu Handle
  MOVE.W #1,-(SP)            ;item num
  NOT.W  Liscio(A5)          ;set & check Smooth flag
  BEQ.S  @0                  ;branch accordingly
  LEA    DrawSmooth(PC),A3   ;routine to call in A3
  PEA    'Unsmooth'          ;load proper name
  BRA.S  @1
@0LEA    DrawPoly(PC),A3           ;no smooth
  PEA    'Smooth'
@1_SetItem
  _OpenRgn
  JSR    (A3)          ;WARNING :(¿) should make sure our poly
is not
  MOVE.L RgnHand(A5),-(SP)   ;too big to fit in a region
  _CloseRgn                  ;if it doesn't fit, we are dead
  BSR    CleanWind
  BSR    ReDraw
  BRA    Loop
; ==============================
DoPensize
  UnChkItm     PenHand,CurPen(A5)  ;uncheck old
  MOVE.W MENU+2(A5),CurPen(A5)     ;v
  MOVE.W MENU+2(A5),CurPen+2(A5)   ;h
  CheckItm     PenHand,CurPen(A5)  ;check new
  TST.W  PolyDrawn(A5)
  BEQ    Loop
  BSR    SetPen
  BSR    CleanWind
  BSR    Redraw
  BRA    Loop
;--------------
DoPat
  UnChkItm     FpatHand,ChkPat(A5) ;uncheck old
  MOVE.W MENU+2(A5),ChkPat(A5)
  CheckItm     FpatHand,ChkPat(A5) ;check new
  MOVE.W MENU+2(A5),D0
  SUBQ.W #1,D0
  ADD.W  D0,D0
  MOVE.W PatTab(PC,D0.W),CurPat(A5) ;get offset constant to
PAT
  TST.W  PolyDrawn(A5)
  BEQ    Loop
  BSR    CleanWind
  BSR    ReDraw
  BRA    Loop
PatTab   DC.W   -8                 ;White
  DC.W   -32                 ;LtGray
  DC.W   -24                 ;Gray
  DC.W   -40                 ;DkGray
  DC.W   -16                 ;Black
; ---------------
DoPenscale
  UnChkItm     PscaleHand,Chkscale(A5)    ;uncheck old
  MOVE.W MENU+2(A5),Chkscale(A5)
  CheckItm     PscaleHand,Chkscale(A5)    ;check new
  MOVE.W MENU+2(A5),D0
  SUBQ.W #1,D0
  ADD.W  D0,D0
  MOVE.W SclTab(PC,D0.W),CurScale(A5) ;get scale factor
  BRA    Loop
SclTab                             ;unsign byte values (0-255)
  DC.W   12                  ;  72/6
  DC.W   25                  ; 150/6
  DC.W   50                  ; 300/6
  DC.W   100                 ; 600/6
  DC.W   200                 ;1200/6
; ==============================
SetPen
  TST.W  WhileDrawing(A5)    ;don't change pen if drawing poly
  BNE.S  @0
  MOVE.L CurPen(A5),-(SP)
  _PenSize
@0RTS
; ---------------
Redraw                             ;no poly to work with
  TST.W  CurPat(A5)          ;is our poly filled ?
  BEQ.S  @0                  ;No fill if CurPat=0
  BSR    FillPoly
@0TST.W  Liscio(A5)          ;is poly smoothed ?
  BNE.S  DrawSmooth
DrawPoly
  MOVE.W NumPoints(A5),D4          ;Total num of points
  SUBQ.W #2,D4               ;sub 1 for DBRA and 1 for _Moveto
  BGT.S  @0            ;check reasonable poly (>2 points)
  RTS                  ;return if not to consider a poly
@0MOVE.L PolyHand,A0   ;No surprise
  _HLock               ;Line and LineTo can cause relocation
  BNE    Error
  MOVE.L (PolyHand),A3       ;Pointer to Poly data
  MOVE.L (A3)+,-(SP)         ;first point
  _MoveTo
NextLine
  TST.W  (A3)                ;Check for bit 15 (sign)
  BPL.S  LongPt              ;IF set do Long ELSE do Short

ShortPt  MOVE.B (A3)+,D1           ;dY: a signed 7 bits
offset
```


© The Definitive MacTutor, Vol. 4

```
  BTST   #6,D1        ;Test sign bit:a negative offset ?
  BNE.S  @0                ;yes, flag bit can stay
  BCLR   #7,D1             ;else (if positive) clear it
@0EXT.W  D1                ;DY as a word
  SWAP   D1                ;in hi word D1 now
  MOVE.B (A3)+,D1          ;signed byte dx
  EXT.W  D1                ;dx in low word
  MOVE.L D1,-(SP)          ;offset on stack
  _Line                    ;draw to
  BRA.S  NxLn

LongPt   MOVE.L (A3)+,-(SP)
  _LineTo                        ;a point
NxLn     DBRA   D4,NextLine

  MOVE.L PolyHand,A0       ;free again
  _HUnLock
  BNE    Error
  RTS
; ---------------
DrawSmooth
  BSR    UnPackPoly        ;we need normal points to work
  MOVE.L PolyBuffer(A5),A0 ;our temporary poly
  _HLock
  BNE    Error
  MOVE.L PolyBuffer(A5),A0
                           ;Emulating a High-Level call
  MOVE.W NumPoints(A5),-(SP) ;First param
  MOVE.L (A0),-(SP)        ;second, a Ptr to points list
  BSR    SmoothPoly        ;fortissimo

  MOVE.L PolyBuffer(A5),A0
  _HUnLock                       ;free again
  BNE    Error
  RTS
;============================================================
; SmoothPoly (NumPt:INTEGER; PtrPolyList:PTR)
; (¿) Implementation good for poly side<1024 pixels of length
; The resulting smoothed polygon closely resembles the laser
output
; ---------------------------
LineTo          EQU    $91       ;trap num
; ---------------
; ------ Stack Frame ------
; ---------------
NumPt           EQU    12
PtrList         EQU    8
ParamBytes      EQU    6
;
A         EQU    -2
B         EQU    -4
Av        EQU    -6
Bv        EQU    -8
X         EQU    -10
Y         EQU    -12
Line            EQU    -16
OldW            EQU    -18
OldQ            EQU    -20

Locals          EQU    -20
; ================================================
; The following is a Basic translation of the algorithm
; Comments in the Asm source refer to these integer variables
; Note that the two version slightly differs where necessary
; P(c,n) is a bidimensional array of points:
; P(0,n)=Y P(1,n)=X ; n=point index
; ---------------
; X1=P(1,1)-P(1,0):Y1=P(0,1)-P(0,0)
; X=X1\2+P(1,0):Y=Y1\2+P(0,0)
; MOVETO X,Y
; FOR j=1 TO NumPt
;    X0=X1:Y0=Y1
;    X1=P(1,j+1)-P(1,j):Y1=P(0,j+1)-P(0,j)
;    T=(ABS(X1+X0)+ABS(Y1+Y0))\8
;    IF T<2 THEN T=2
;    Z=T*T*2:Du=T+1
;   FOR n=0 TO T
; Du=Du-1:Q=n*n:w=Du*Du
; A=(Q*X1-w*X0)/Z+P(1,j)
; B=(Q*Y1-w*Y0)/Z+P(0,j)
; LINETO A,B
;   NEXT
; NEXT
; LINETO X,Y
; ================================
SmoothPoly               ;If called from hi level think of it
  LINK   A6,#Locals      ;like a ToolBox procedure
  MOVEM.L      A1-A4/D0-D7,-(SP)   ;save the world

  MOVE.L PtrList(A6),A4            ;A4 point list Ptr
  MOVE.W #LineTo,D0        ;Trap num for LineTo
  _GetTrapAddress                  ;save  some cycle
  MOVE.L A0,A3             ;store in A3 for later use
  CLR.L  Bv(A6)            ;clear last drawn point

  MOVE.L (A4),D4                   ;First Pt
  MOVE.L 4(A4),D5          ;second Pt
  SUB.W  D4,D5             ;X1
  MOVE.W D5,D1             ;safe in D1
  SWAP   D4                ;push y in low word
  SWAP   D5
  SUB.W  D4,D5             ;Y1
  MOVE.W D5,D0             ;safe in D0
  ASR.W  #1,D1             ;divs by 2 with no sign loss
  ASR.W  #1,D0
  ADD.W  D4,D0             ;D0=Y+(Y2-Y1)/2
  ADD.W  2(A4),D1          ;D1=X+(X2-X1)/2
  MOVE.W D0,Y(A6)          ;First point recorded for closing
  MOVE.W D1,X(A6)
  MOVE.W D1,-(SP)
  MOVE.W D0,-(SP)
  _MoveTo
  SUBQ.W #1,NumPt(A6)      ;ready to start
NextPoint
  MOVE.L D5,D4             ;Y0=Y1 : X0=X1
  ADDQ.L #4,A4             ;Current=Next
  MOVE.L 4(A4),D5          ;Next Point in D5
  SUB.W  2(A4),D5          ;X1=NextX-CurX
  MOVE.W D5,D6             ;safe in D6
  SWAP   D5                ;push Y down
  SUB.W  (A4),D5                   ;Y1=NextY-CurY
  MOVE.W D5,D0             ;safe in D0
;                    Calc num iteration now (T) ------
  ADD.W  D4,D0             ;D0=Y0+Y1
  BPL.S  @1
  NEG.W  D0                ;ABS
@1SWAP   D4                ;x in low word
  ADD.W  D4,D6             ;D6=X0+X1
  BPL.S  @2
  NEG.W  D6                ;ABS
@2SWAP   D4                ;y back in low

  ADD.W  D0,D6             ;D6=(D0^2+D6^2)
  LSR.W  #3,D6             ;T=D6\8 (step)
  CMP.W  #2,D6             ;T=>2 at least
  BPL.S  @0
  MOVEQ  #2,D6             ;T=D6
@0MOVE.W D6,D7
  MULU   D7,D7             ;T*T
  LSL.W  #1,D7             ;Z=D7 (T*T*2)
  MOVEQ  #0,D3             ;n=0

  MOVE.W D6,D0             ;init values
  ADDQ.W #1,D0
  MULU   D0,D0
  MOVE.W D0,OldW(A6)
  MOVE.W #1,OldQ(A6)
Curva
```

```
  MOVE.W D6,D0              ;Calc W (T*T), avoiding slow MULU
  ADD.W  D0,D0
  ADDQ.W #1,D0
  SUB.W  D0,OldW(A6)        ;W=(OldW)-2T+1
  MOVE.W OldW(A6),D0

  MOVE.W D3,D1              ;Calc Q (n*n), avoiding slow MULU
  ADD.W  D1,D1
  SUBQ.W #1,D1
  ADD.W  D1,OldQ(A6)        ;Q=(OldQ)+2n-1
  MOVE.W OldQ(A6),D1

  ADDQ.W #1,D3              ;n=n+1

  MOVE.W D0,D2              ;Calc B ——
  MULS   D4,D2              ;Y0*W
  MOVE.L D2,A2
  MOVE.W D1,D2
  MULS   D5,D2              ;Y1*Q
  SUB.L  A2,D2
  DIVS   D7,D2              ;/Z
  ADD.W  (A4),D2                ;+ CurPt Y
  MOVE.W D2,B(A6)           ;safe

  SWAP   D4                 ;push Xs down
  SWAP   D5

  MOVE.W D0,D2              ;Calc A ——
  MULS   D4,D2              ;X0*W
  MOVE.L D2,A2
  MOVE.W D1,D2
  MULS   D5,D2              ;X1*Q
  SUB.L  A2,D2
  DIVS   D7,D2              ;/Z
  ADD.W  2(A4),D2           ;+ CurPt X
  MOVE.W D2,A(A6)           ;save
  SWAP   D4
  SWAP   D5

  CMP.W  Av(A6),D2    ;did we move? (over the unit)
  BNE.S  @0           ;much faster to check it here, than go
  MOVE.W B(A6),D2     ;through the ROM with a useless LineTo
  CMP.W  Bv(A6),D2
  BEQ.S  @1

@0MOVE.L B(A6),-(SP)
  JSR    (A3)               ;LineTo
  MOVE.L B(A6),Bv(A6)
@1DBRA   D6,Curve

  SUBQ.W #1,NumPt(A6)
  BGT    NextPoint

Done    MOVE.L Y(A6),-(SP)          ;back to first point
  JSR    (A3)               ;LineTo

  MOVEM.L      (SP)+,A1-A4/D0-D7   ;restore the world
  UNLK   A6
  MOVE.L (SP)+,A0           ;RTS
  ADDQ   #ParamBytes,SP
  JMP    (A0)               ;back home
; ——————————
UnPackPoly
  MOVE.L PolyBuffer(A5),A0  ;copy the poly here
  MOVE.L (A0),A0                ;Ptr to temporary space
  MOVE.W NumPoints(A5),D0
  MOVE.W BytesWritten(A5),D1
  MOVE.L (PolyHand),A1      ;Ptr to packed poly in A1
  MOVE.L (A1)+,(A0)+        ;first point
  SUBQ.W #1,D0              ;one pt processed
  MOVE.L (A1),-4(A1,D1.W)   ;add 2nd pt to the end of list
Spacca  TST.W  (A1)         ;Check if short or long point
  BMI.S  @0
  MOVE.L (A1)+,(A0)+        ;nothing to unpack
  DBRA   D0,Spacca
  BRA.S  SpacDon
@0MOVE.L -4(A0),(A0)        ;point to add the offset to
  MOVE.B (A1)+,D1           ;dY: a signed 7 bits offset
  BTST   #6,D1              ;a negative offset ?
  BNE.S  @1                 ;yes, flag bit can stay
  BCLR   #7,D1              ;positive: clear flag bit
@1EXT.W  D1                 ;DY as a word
  ADD.W  D1,(A0)+
  MOVE.B (A1)+,D1           ;signed byte dx
  EXT.W  D1                 ;dx in low word
  ADD.W  D1,(A0)+
  DBRA   D0,Spacca
SpacDon  RTS
; ====================================================
FillPoly
  MOVE.L RgnHand(A5),-(SP)  ;Our Handle
  MOVE.W CurPat(A5),D0      ;Offset to pat
  MOVE.L (A5),A0                ;Ptr to QD's GlobalsPEA
0(A0,D0.W)          ;fill it with Current Pat
  _FillRgn
  RTS
; ——————————
CleanWind
  MOVE.L TheWind(A5),A0
  PEA    PortRect(A0)
  _EraseRect
  BSR    PutGrow                ;show it
  RTS
; ——————————
ClipWind
  MOVE.L TheWind(A5),A0
  PEA    PortRect(A0)
  _ClipRect
  RTS
; ——————————
PutGrow
  MOVE.L #$10001,-(SP)
  _PenSize                      ;Use pen 1,1 to draw
  MOVE.L TempRgn(A5),-(SP)  ;save
  _GetClip
  PEA    GrowBox(A5)        ;only the box
  _ClipRect
  MOVE.L TheWind(A5),-(SP)
  _DrawGrowIcon
  MOVE.L TempRgn(A5),-(SP)  ;put back
  _SetClip
  BSR    SetPen             ;restore pen
  RTS
; ——————————
CalcGrow
  MOVE.L TheWind(A5),A0
  LEA    GrowBox(A5),A1         ;Recalculate new GrowBox
  MOVE.L PortRect+4(A0),(A1) ;push bottom right point of
window
  SUB.W  #15,(A1)+          ;make Top left box
  SUB.W  #15,(A1)+
  MOVE.L PortRect+4(A0),(A1) ;and bottom right
  RTS
; ——————————
ChkDoubClick                    ;Check if a double click
  CLR.W  DoubleClick(A5)        ;assume not
  MOVE.L WHEN(A5),D1        ;this click
  SUB.L  LastClick_T(A5),D1 ;ticks elapsed since last
  CMP.L  DoubleTime,D1      ;close enough together?
  BGT.S  @0                 ;branch if over 'DoubleTime'
  MOVE.W WHERE(A5),D0       ;now check for position
  SUB.W  LastClick_W(A5),D0 ;delta Y
  BPL.S  @1                 ;ABS
  NEG.W  D0
@1MOVE.W WHERE+2(A5),D1         ;delta X
  SUB.W  LastClick_W+2(A5),D1
  BPL.S  @2                 ;ABS
  NEG.W  D1
```


© The Definitive MacTutor, Vol. 4

```
@2ADD.W  D0,D1                ;Dx+Dy
  CMP.W  #3,D1                ;allow for a 3 pix max distance
  BGT.S  @0
  ST     DoubleClick(A5)            ;yes, it's a double click
@0MOVE.L WHEN(A5),LastClick_T(A5);Record time and…
  MOVE.L WHERE(A5),LastClick_W(A5) ;location
  RTS
; ——————————————
ActMenu  BSR    paramOnStack        ;load the stack
  _EnableItem                 ;3 items
  _EnableItem                 ;to enable
  _EnableItem
  _DrawMenuBar
  RTS
DimMenu  BSR    paramOnStack        ;load the stack
  _DisableItem                ;3 items
  _DisableItem                ;to disable
  _DisableItem
  _DrawMenuBar
  RTS
; ——
paramonStack
  MOVE.L (SP)+,A0             ;RTS in A0
  MOVE.L EditHand(A5),-(SP)
  MOVE.W #1,-(SP)             ;undo
  MOVE.L EditHand(A5),-(SP)
  MOVE.W #3,-(SP)             ;cut
  MOVE.L EditHand(A5),-(SP)
  MOVE.W #5,-(SP)             ;paste
  JMP    (A0)
; ========================================
MakePoly
  ShortComment PolyBegin      ;Here starts our special Poly
  ShortComment PicPlyClo      ;always a closed Poly in our case

  TST.W  Liscio(A5)           ;is our poly to be smoothed?
  BEQ.S  @1                   ;branch if not

  MOVE.L CommentHand(A5),A1   ;Handle to smooth verbs
  MOVE.L (A1),A0              ;deference
  MOVEQ  #Frame+Close,D0      ;assume only closed and framed
  TST.W  CurPat(A5)           ;is our poly filled ?
  BEQ.S  @0
  ADDQ.W #Fill,D0             ;add fill verb
@0MOVE.B D0,(A0)              ;set verb = Close+Frame+(Fill)
  LongComment  PolySmooth,1,A1      ;smooth flag true

  MOVE.L TempRgn(A5),-(SP)    ;save clip
  _GetClip
  PEA    ZeroRect(PC) ;if not aware of comments then no print
  _ClipRect
  BSR    DrawPoly;Pass laser original vertexes to be smoothed
  MOVE.L TempRgn(A5),-(SP)
  _SetClip                    ;allow others to print

  ShortComment PolyIgnore     ;ignore following poly if laser

@1BSR    ReDraw       ;simulate spline for non-laser printers
  ShortComment PolyEnd              ;End special poly
  RTS
;========================================
InitGlobals
  CLR.W  CurPat(A5)           ;No fill pat
  MOVE.L #$10001,CurPen(A5)   ;PenSize 1,1
  MOVE.W #12,CurScale(A5)     ;72 dpi
  MOVE.W #1,ChkPat(A5)        ;first pat checked
  MOVE.W #1,Chkscale(A5)      ;72 dpi checked
  RTS
; ——————————————
InitMac
  PEA    -4(A5)               ;Recite Mac pray
  _InitGraf
  _InitFonts
  _InitWindows
  _InitMenus
  CLR.L  -(SP)                ; No restart procedure
  _InitDialogs
  _TEInit
  _InitCursor
  MOVE.L #$0000FFFF,D0        ; Flush all events
  _FlushEvents
  RTS
; ——————————————
SetWindow
  MOVE.L (A5),A0              ;Pointer to QuickDraw Globals
  LEA    screenBits(A0),A0    ;Ptr to Bounds of Screen BitMap
  MOVE.L 6(A0),WRect(A5)             ;Get & set Screen Rect
  MOVE.L 10(A0),WRect+4(A5)
  PEA    WRect(A5)
  MOVE.L #$100010,-(SP)              ;allow 16 Pixels inside
  _InsetRect
  MOVE.L WRect(A5),WBounds(A5)       ;Init Draging bounds Rect
  MOVE.L WRect+4(A5),WBounds+4(A5)

  MOVE.L #$400080,WSize(A5)  ;Grow Rect (64,128,768,768)
MOVE.L   #$3000300,WSize+4(A5)      ;(¿) should calc this rect
                              ;according to PageSetUp
  MOVE.W MBarHeight,D0        ;Center WRect on video
  ADDI.W #18,D0               ;Title bar
  ADD.W  D0,WRect(A5)         ;subtract Menu Height

  CLR.L  -(SP)                ;Create our window
  CLR.L  -(SP)
  PEA    WRect(A5)
  PEA    'Polygon'            ;document:Polygon
  MOVE.B #-1,-(SP)
  MOVE.W #8,-(SP)             ;has zoom box
  MOVE.L #-1,-(SP)
  CLR.B  -(SP)
  CLR.L  -(SP)
  _NewWindow
  MOVE.L (SP),TheWind(A5)     ;leave it on stack and save
  _SetPort
  BSR    CalcGrow
  BSR    ClipWind
  BSR    PutGrow
  RTS
;================================
SetMenus
  MOVEM.L      A3/D3-D5,-(SP)            ;save stuff
  LEA    MenuHndList(A5),A3  ;Stuff with MHandles from here on
  MOVEQ  #1,D3               ;Start with Menu 1
  MOVEQ  #4-1,D4             ;4 menus (-1 for DBRA)
  MOVEQ  #0,D5               ;before ID = 0
  BSR    PushMenus           ;sequential ordering

  MOVE.L MelaHand(A5),-(SP)
  MOVE.L #'DRVR',-(SP)
  _AddResMenu                ;yes, we support DA
                             ;add submenus
  MOVEQ  #'A',D3             ;Start with Menu 65
  MOVEQ  #2,D4               ;through menu 67
  MOVEQ  #-1,D5              ;hierarchical beforeID
  BSR    PushMenus

  CheckItm     PenHand,#1          ;check items
  CheckItm     FpatHand,#1
  CheckItm     PscaleHand,#1

  _DrawMenuBar
  MOVEM.L      (SP)+,A3/D3-D5      ;restore stuff
  RTS
; ————————
PushMenus
  CLR.L  -(SP)
  MOVE.W D3,-(SP)             ;ID of Menu to get
  _GetRMenu
  MOVE.L (SP),(A3)+    ;Save Hndl trusting global ordering
```

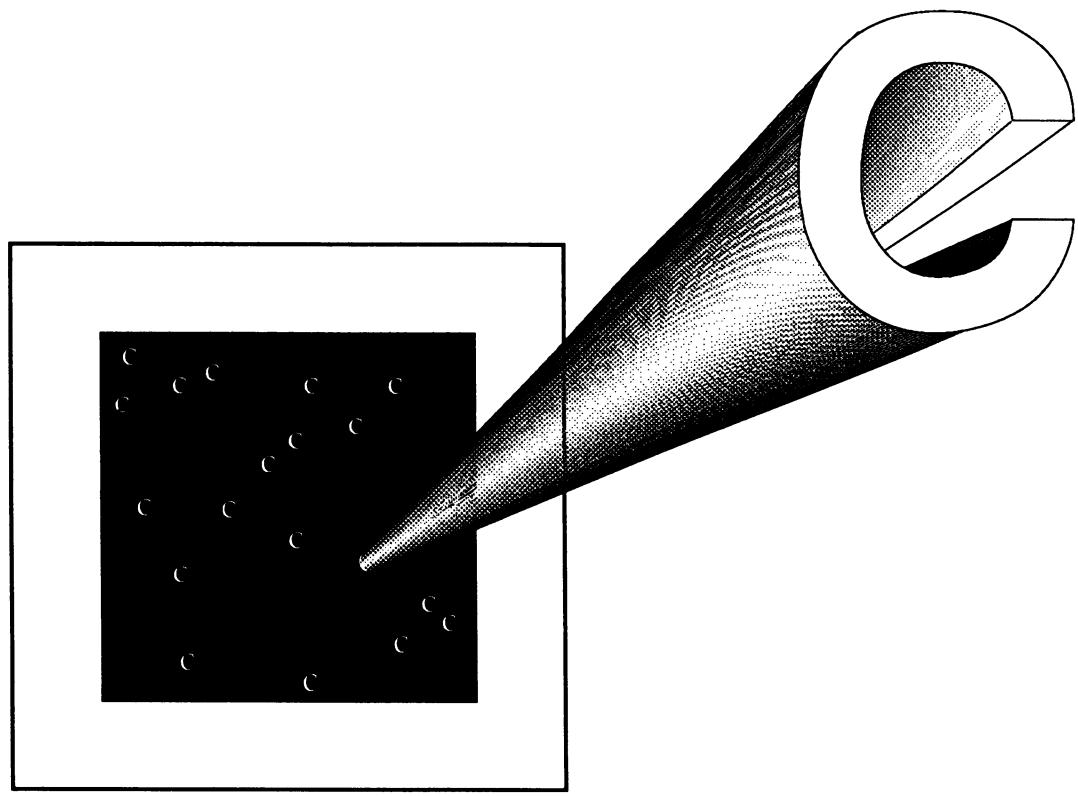
```
  MOVE.W D5,-(SP)     ;Hndl was left on stack + beforeID
  _InsertMenu
  ADDQ.W #1,D3              ;Next ID
  DBRA   D4,PushMenus
  RTS
; ————————————
GetHandles
  MOVE.L #$7FFF,D0          ;ask for 32K
  _NewHandle
  BNE    Error
  MOVE.L A0,PolyHand

  MOVE.L #$FFFE,D0          ;ask for 64K
  _NewHandle
  BNE    Error
  MOVE.L A0,PolyBuffer(A5)

  MOVEQ  #120,D0                  ;Print Record Size
  _NewHandle
  MOVE.L A0,PrintRec(A5)
  MOVE.L A0,-(SP)
  prExec PrintDefault       ;Fill it with defaults

  MOVEQ  #8,D0              ;enough for any comment data
  _NewHandle
  BNE    Error
  MOVE.L A0,CommentHand(A5)

  CLR.L  -(SP)              ;our fill region
  _NewRgn
  MOVE.L (SP)+,RgnHand(A5)

  CLR.L  -(SP)
  _NewRgn
  MOVE.L (SP)+,TempRgn(A5)  ;A dummy region
  RTS
;=======================================
ERROR    BSR    Bippa       ;WARNING (¿)
  BSR    Bippa       ;Best Error Handler routine ever written
  BSR    Bippa       ;but I know you could do even better
  _ExitToShell
; ————————————
Bippa    MOVE.W #3,-(SP)
  _SysBeep
  RTS
; ————————————
OroCurs  CLR.L  -(SP)       ;watch cursor
  MOVE.W #4,-(SP)
  _GetCursor
  MOVE.L (SP)+,A0
  MOVE.L (A0),-(SP)
  _SetCursor
  RTS
; ————————————————————
ZeroRect DC.W   0,0,0,0
; ————————————————————
; ———————— GLOBALS ————————
; ————————————————————
WRect           DS.B   8      ;Window stuff
GrowBox         DS.B   8
WBounds         DS.B   8
WSize           DS.B   8
TheWind         DS.L   1
WINDOW          DS.L   1

MenuHndList     DS.L   0      ;do not change this ordering!
MelaHand        DS.L   1      ;Menu Handles
FileHand        DS.L   1
EditHand        DS.L   1
DoHand          DS.L   1
PenHand         DS.L   1      ;Hierarchical Menu Handles
FpatHand        DS.L   1
PscaleHand      DS.L   1

EVENTRECORD     DS.B   0      ;My event.
WHAT            DS.W   1
MESSAGE         DS.L   1
WHEN            DS.L   1
WHERE           DS.L   1
MODIFY          DS.W   1

MENU            DS.L   1      ;Other…
PolyBuffer      DS.L   1
PolyDrawn       DS.W   1
whileDrawing    DS.W   1
NumPoints       DS.W   1
BytesWritten    DS.W   1
LastPoint       DS.L   1
CurPoint        DS.L   1
OldPoint        DS.L   1
MousLoc         DS.L   1
LastClick_T     DS.L   1
LastClick_W     DS.L   1
DoubleClick     DS.W   1
CurPat          DS.W   1
CurPen          DS.L   1
CurScale        DS.W   1
ChkPat          DS.W   1
Chkscale        DS.W   1
Liscio          DS.W   1
RgnHand         DS.L   1
TempRgn         DS.L   1
CommentHand     DS.L   1
DskName         DS.B   64
SavePort        DS.L   1
PrintPort       DS.L   1
PrintRec        DS.L   1
PrStatus        DS.B   26
ItmHit          DS.W   1
```

Listing: Smooth.R

```
* Resources for Smoother
* (Make sure blank lines don't have space characters in them!)

SmoothRes.Rel
APPLDEMO

TYPE DEMO = GNRL
,0
.P
v 1.0 by Raul Tabasso - Roma 22/6/88

TYPE FREF
   ,128
APPL 0

TYPE BNDL
   ,128
DEMO 0
ICN#
0 128
FREF
0 128

TYPE ICN# = GNRL
   ;; Appl Icon
,128
.H
0000 0000 0000 0000 0000 0000 0000 3FC0
0001 C030 0002 0008 0004 0004 0004 0002
0008 0002 0008 0002 0008 0182 0008 0204
0004 0208 0002 01F0 0001 0000 0000 8000
0000 6000 0000 1800 0000 0700 0000 00C0
0000 0030 0000 000C 0000 0004 0000 0002
0000 0002 0000 0002 0000 0002 0000 0004
0000 0004 0000 0008 0000 0030 1FFF FFC0
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
```


© The Definitive MacTutor, Vol. 4

```
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
* ————————————————
* - MENUS

TYPE MENU
  ,1;;Apple menu
\14   ;;Apple char
About Smoother…
(-

TYPE MENU
  ,2
File  ;;File menu
New polygon/N
Page Setup…
(Print…
Laser pen scaling/\1B!C
;;submenu 67
Quit/Q

TYPE MENU
  ,3
Edit  ;;Edit menu
Undo/Z
(-
Cut/X
Copy/C
Paste/V

TYPE MENU
  ,4
Options    ;;Polygon menu
(Smooth/S
Pen size/\1B!A ;;submenu 65
Fill tone/\1B!B ;;and 66

TYPE MENU
  ,65
Psize    ;; Pen Size submenu
1 Pt.^;
2 Pt.^<
3 Pt.^=
4 Pt.^>
5 Pt.^?

TYPE MENU
  ,66
Fpat  ;; Fill Pattern submenu
No fill^1
25%^2
50%^3
75%^4
Black^5

TYPE MENU
  ,67
PScale   ;; Pen Scaling submenu
72 dpi
150 dpi
300 dpi
600 dpi
1200 dpi

TYPE DLOG
  ,256;;ID
The about dialog
80 80 220 400;;Rect
Visible NoGoAway
1   ;;proc ID
0   ;;ref con
256   ;;DITL related

TYPE DITL
  ,256
2

Button
100 200 120 256
OK

StaticText
32 64 80 256
Smoother by Raul Tabasso. ++
An example program on polygon's comments

* ————————————————
* - Icons for submenu n° 66 ———

TYPE ICON = GNRL
  ,257
.H
FFFF FFFF FFFF FFFF C000 0003 C000 0003
C000 0003 C000 0003 C000 0003 C000 0003
C000 0003 C000 0003 C000 0003 C082 3C03
C0C2 6603 C0E2 6603 C0F2 6603 C0BA 6603
C09E 6603 C08E 6603 C086 6603 C082 3C03
C000 0003 C000 0003 C000 0003 C000 0003
C000 0003 C000 0003 C000 0003 C000 0003
C000 0003 C000 0003 FFFF FFFF FFFF FFFF

TYPE ICON = GNRL
  ,258
.H
FFFF FFFF FFFF FFFF E222 2223 C000 0003
C888 888B C000 0003 E222 2223 C000 0003
C888 888B C000 0003 E222 2223 C000 0003
C888 888B C000 0003 E222 2223 C000 0003
C888 888B C000 0003 E222 2223 C000 0003
C888 888B C000 0003 E222 2223 C000 0003
C888 888B C000 0003 FFFF FFFF FFFF FFFF

TYPE ICON = GNRL
  ,259
.H
FFFF FFFF FFFF FFFF EAAA AAAB D555 5557
EAAA AAAB D555 5557 EAAA AAAB D555 5557
EAAA AAAB D555 5557 EAAA AAAB D555 5557
EAAA AAAB D555 5557 EAAA AAAB D555 5557
EAAA AAAB D555 5557 EAAA AAAB D555 5557
EAAA AAAB D555 5557 EAAA AAAB D555 5557
EAAA AAAB D555 5557 FFFF FFFF FFFF FFFF

TYPE ICON = GNRL
  ,260
.H
FFFF FFFF FFFF FFFF DDDD DDDF F777 7777
DDDD DDDF F777 7777 DDDD DDDF F777 7777
DDDD DDDF F777 7777 DDDD DDDF F777 7777
DDDD DDDF F777 7777 DDDD DDDF F777 7777
DDDD DDDF F777 7777 DDDD DDDF F777 7777
DDDD DDDF F777 7777 DDDD DDDF F777 7777
DDDD DDDF F777 7777 FFFF FFFF FFFF FFFF

TYPE ICON = GNRL
  ,261
.H
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF

* ————————————————
* - Icons for submenu n° 65 ———

TYPE ICON = GNRL
  ,267
.H
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 FFFF FFFC
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000

TYPE ICON = GNRL
  ,268
.H
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 FFFF FFFC FFFF FFFC
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000

TYPE ICON = GNRL
  ,269
.H
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 FFFF FFFC FFFF FFFC FFFF FFFC
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000

TYPE ICON = GNRL
  ,270
.H
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
FFFF FFFC FFFF FFFC FFFF FFFC FFFF FFFC
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000

TYPE ICON = GNRL
  ,271
.H
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 FFFF FFFC
FFFF FFFC FFFF FFFC FFFF FFFC FFFF FFFC
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
```



C Workshop

© The Definitive MacTutor, Vol. 4

ABC's of C

Breaking the Four Window Barrier

William Rausch
Kennewick, WA
MacTutor Vol. 4 No. 1

It seems that every good Macintosh example program (of a text editor, graphics program, or whatever) is hung up on the idea of four windows. Of course, those that aren't such good example programs only use one window! This limit always seems to be a result of the unnecessary use of global data structures.

Most multi-window Macintosh example programs contain some lines that look like:

```
WindowPtr some_windows[how_many];
data_structure some_data[how_many];
```

where how_many is defined to be four. The programmer will claim that it is easy to change this number to be as large as necessary, and the application can be easily recompiled. That is fine for the programmer, but what about the end user. He is stuck with whatever number the programmer happens to choose. If the programmer happens to choose a number of windows and data structures that perfectly fits a 512K Macintosh, then the user is probably not able to take advantage of the extra memory of a Macintosh Plus, much less a Macintosh with even more memory.

In actuality, it is not necessary to predefine any number of windows, or the data structures that are associated with them. Since the Macintosh has an operating system that knows all about windows and how to keep track of them, it makes little sense for the application programmer to duplicate this effort.

The window data structure contains a 32-bit field called refCon which Inside Macintosh describes as "the window's reference value field, which the application may store into and access for any purpose". One such purpose, which is mentioned but not demonstrated in many books on Macintosh programming, is to store a handle to some application specific data structure which contains the information that is pertinent to that window.

Since every event given to an application by the Event Manager will contain a pointer to the window concerned, there is no need for the application to keep an array of WindowPtrs; and since each window contains a refCon field, there is no need to keep an array of data structures. Instead, allocate the necessary space for the data structure at the time the new window is created, and put its handle in the refCon field of the window's data structure. Then, whenever an event is processed, just pass the WindowPtr along to the functions which are going to process the event. These functions then can get access to the data structure when they need it.

The code fragments which follow demonstrate the concepts described:

```
enum window_type
  (
  type_a,
  type_b,
  type_c
  );

typedef union
  (
  struct
    (
    ...          /* type_a window data */
    ) a;
  struct
    (
    ...          /* type_b window data */
    ) b;
  struct
    (
    ...          /* type_c window data */
    ) c;
  ) content,*content_ptr,**content_hndl;

typedef struct
  (
  enum window_type xxx;   /* optional, appl. specific */
  content_hndl yyy;   /* could include union directly */
  ...
  ) window_data, *data_ptr, **data_hndl;

...

make_window(type)
enum window_type type;
  (
  WindowRecord *a_window;
  data_hndl a_data;
  content_hndl a_content;
  ...

  a_window =
    (WindowRecord *)NewWindow(...);
  a_data =
    (data_hndl)NewHandle(sizeof(w_data));
  a_content = (content_hndl)NewHandle
                    (sizeof(content));
  (**a_data).xxx = type;
  (**a_data).yyy = a_content;
  a_window->refCon = (long)a_data;
  ...
  )

...

event_loop()
  (
  EventRecord the_event;
  WindowPtr which_window;
  int window_code;
  ...

  while (TRUE)
    (
    if (GetNextEvent(everyEvent,
                     &the_event))
```

```
    {
    switch (the_event.what)
      {
      case mouseDown:
      window_code = FindWindow
                    (the_event.where,
                     &which_window);
      switch (window_code)
        {
        case inSysWindow:
        SystemClick(&the_event,
                    which_window);
        break;

        case inContent:
        process_window(which_window);
        break;

        ...
        }
      break;

      ...
      }
    }
  else
    {
    }
  }
}

...

process_window(the_window)
WindowRecord *the_window;
  {
  data_hndl the_data;
  ...

  the_data =
      (data_hndl)(the_window->refCon);
  switch ((**the_data).xxx)
    {
    case type_a:
    process_a_window(the_window);
    break;

    case type_b:
    process_b_window(the_window);
    break;

    case type_c:
    process_c_window(the_window);
    break;
    }
  ...
}
```

This general technique may, in some very particular cases, slow things down a little. Generally, this won't be the case because most applications spend a majority of their time processing NULL events. If you wanted to speed things up a little bit and still maintain the dynamic allocation capabilities of this programming approach, use just one globally declared WindowPtr and one data_handle and just update them to always point at the window/data_structure that the current event refers to. Then you wouldn't have to pass the event's WindowPtr to the called function(s).

Of course, most applications need to be able to find all of their windows at times, e.g., when Quitting the application so that changes can be saved, or when updating the windows. At those times, it is still possible to avoid using a global array merely by calling FrontWindow() and following the linked list of pointers stored in the nextWindow field of the window's data structure.

This approach to creating windows in an application, which I call "fire and forget" (and which could be considered as a small first step towards object oriented programming), will allow your application to create as many new windows (and their associated data structures) as the user desires. It is also a convenient way of manipulating different types of windows in a single application. (It is very easy, for example, to imagine MacDraw and MacWrite in a single application!) Of course, your application has to check available memory each time it tries to create a new window/data structure to make sure there is enough available, but that is a small price to pay for the increased utility the user can get from having a program that "grows" with his hardware.

Postscript Programming
Overview & Downloader in C

Nicholas Pavkovic
Perimeter
Chicago, IL
MacTutor Vol. 1 No. 1

In this article, I'm going to present a programmer's overview of PostScript. This overview includes comparisons between PostScript and QuickDraw, information about using the stack and a brief explanation of the LaserWriter driver. At the end of the article, I'll present the PAP manager calls (which allow direct access to the LaserWriter over AppleTalk) and C source code for a PostScript downloader.

Overview of PostScript

PostScript is a graphics language that's designed to set type and graphics on high-resolution devices such as laser printers and typesetting machines. In most of its implementations it runs on a dedicated processor that's contained in or connected to a printer. Though these printers vary considerably in their capabilities, PostScript is device-independent, so programs written for one PostScript device will usually run on another.

In addition to its graphic capabilities, PostScript contains a full complement of standard language features (floating point arithmetic, string and array manipulation, etc.) and could conceivably be used for non-graphics applications. It most closely resembles FORTH in its syntax and overall "feel" (but, thankfully, PostScript carries no social stigma). However, most FORTH systems provide the option of controlling the processor at a very low level and PostScript (because it is device-independent) allows only high-level programming.

PostScript's design shows the influence of SmallTalk. Though it isn't really object-oriented (there is no messaging system and no inheritance), the documentation describes instances of the data types as "objects," and the interpreter performs comprehensive type checking. The influence of SmallTalk can probably be traced to PARC, where PostScript's principal designer, John Warnock, worked during the late 70's and early 80's. Since SmallTalk and QuickDraw share common ground, Mac programmers will recognize some PostScript commands.

Terminology

In the PostScript documentation, the built-in commands are usually referred to as operators. In this article, they will be referred to in a number of more familiar ways, such as routines, procedures, functions, etc. The name used doesn't provide any additional information about the operator: an operator referred to as a function won't necessarily return a value. The data used by operators are technically called operands, but I often use arguments and parameters instead. It's good to use the correct PostScript terminology, but unfamiliar terms can make a new language seem even more foreign.

PostScript and QuickDraw

To give you a better sense of PostScript's capabilities, I'd like to cover some of the similarities and differences between PostScript and QuickDraw. While reading this section, think of the PostScript function calls as Pascal or C calls in which the arguments precede the routine names. Note that the PostScript interpreter is case-sensitive and all of the built-in commands are written with lowercase letters.

Coordinates, Navigation and the Path

Unlike QuickDraw, PostScript uses a Cartesian coordinate system with the origin located in the lower left hand corner of the page. The x coordinates increase from left to right and the y coordinates increase from bottom to top. In this coordinate system, the basic unit is the point which is 1/72" wide (the width of a pixel on the Macintosh screen). Since PostScript supports floating point, coordinates don't have to be integers.

Like QuickDraw, PostScript uses a pen and positions it with the moveto command. Since the arguments precede the routine, the code to move the pen to position (72, 18) would read 72 18 moveto. To move the pen relative to its current position, use rmoveto. Straight line segments are drawn using lineto and rlineto. These commands are preceded by an x and y coordinate (or offset, if relative movement is involved).

Unlike QuickDraw, line segments are not drawn on the page as the commands are executed. The segments are accumulated into the current path, a hidden data structure that keeps track of the lines drawn on the page. Once you've created a path, you can draw (outline) it using the stroke operator or fill its interior using the fill operator (neither operator takes any arguments). The nature of the fill or the outline depend on the pen's characteristics when fill or stroke is called. The PostScript pen's state differs somewhat from the QuickDraw pen. The PostScript pen does not have a transfer mode; whatever you draw will cover anything printed underneath it. And though the PostScript pen can be set up to draw with a pattern, it's a somewhat tricky procedure. Usually you'll use the PostScript operator setgray, which allows you to fill the path with different levels of gray. setgray is preceded by the gray level, a decimal that ranges from 0 (black) to 1 (white).

PostScript's setlinewidth is the PostScript version of QuickDraw's PenSize routine; it's preceded by the desired line weight (vertical width), which can be fractional. This command doesn't alter the horizontal width of the pen. 0 setlinewidth generates the thinnest line available on the PostScript device you're using. Lines are drawn from the center; half of the line's weight is above the segment specified and half is below.

fill and stroke have one surprising side-effect: they erase the current path and make the current pen position indefinite. Be-

cause the pen position is indefinite, you'll need a moveto to position the pen after you've used one of these operators. The erasure of the current path causes problems if you want to create a figure that's both stroked and filled. To do so, you must either draw the object twice (fill it once and stroke it the second time) or use the commands gsave and grestore. These commands save and restore the graphics state, which includes the current path and pen state (among other things). gsave fill grestore stroke fills and strokes the current path.

Once you've finished drawing and you want to see your creation, execute showpage. Up to this point, all drawing has been done on a bit image of the page that exists only in memory. showpage transfers the memory image to the physical page and ejects the page.

Setting Text

Before you can draw any text, you must indicate the font that you wish to use. The commands needed to set the font to 18 point Times-Roman are:

```
/Times-Roman findfont
12 scalefont
setfont
```

It's a little too early to explain exactly why this works, but I will point out that findfont can be preceded by any valid font name (beginning with a backslash) and scalefont is preceded by the desired font size. If you don't know the correct spellings of the font names, you can find them by executing FontDirectory pop = forall, another piece of code that will have to remain unexplained. Note that PostScript uses a different name for each font style, and certain standard Macintosh styles (Outline, Shadow and Underline) aren't standard on the LaserWriter.

After you've set the font, you'll need to move into position with moveto. Then use show to draw the text. show is preceded by the text to be drawn, enclosed in parentheses, as in (Hello, world) show. The current gray level will be used to draw the text.

Coordinate Transformations

Before being drawn, every coordinate is transformed by the current transformation matrix or CTM. To modify the drawing space, you can adjust the CTM directly with some general matrix operators, or you can avoid the linear algebra and use the following "convenience" operators:

x y scale	Scales the drawing space Default: 1 1 scale
rotate	Rotates the axes
x y translate	Translates the origin to the point (x,y)

Note that these operators affect drawing that is done after they are invoked; they don't affect any text or graphics that have already been drawn.

What's Missing?

There are a few features that QuickDraw programmers will miss in PostScript.

As mentioned above, there are no transfer modes; you'll have to be careful about the order in which objects are drawn and make use of PostScript's clip operator.

Though PostScript's text processing is more sophisticated than QuickDraw's, PostScript contains no operators for wrapping text (there's no TextBox equivalent). You can write a simple line-breaking procedure in PostScript, but for reasons of speed you'll probably want to do any heavy-duty hyphenation and justification on the Mac's processor.

Regions are missing from PostScript, as are most of QuickDraw's routines for drawing geometric shapes. In both cases, it's not difficult to program functional equivalents.

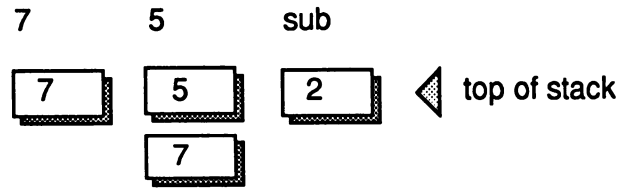
Visualizing the Stack

Like FORTH, PostScript is a stack-based language. FORTH programmers will find their experience with FORTH's stack very useful when learning to use PostScript.

Assembly language programmers will also benefit from their familiarity with the stack. The stack preparation required by PostScript procedures is conceptually similar to the setup that occurs in assembler prior calling a Pascal function or on entry to a definition routine that follows the Pascal calling conventions. In most respects, however, PostScript and assembler are quite different. The PostScript stack is "intelligent" in the sense that it keeps track of its elements' types and sizes. PostScript relies on it for arithmetic operations that assembler normally handles with registers, and PostScript provides a full set of methods for changing the order of elements on the stack. One thing that PostScript doesn't use its operand stack for is return addresses, so you don't have to worry about accidently branching into a data structure if your calls aren't set up correctly.

For programmers used to high-level languages, the stack will seem somewhat cumbersome at first. For example, subtracting 5 from 7 is accomplished with the code 7 5 sub. *[If you remember the great HP 65 calculator and it's reverse polish notation, then this will appeal to you. -Ed]* As each number is read, it's pushed onto the top of the stack. The prior contents of the stack are moved downwards. Keeping in mind that PostScript is interpreting the statement from left to right, consider the following visual representation of what happens:

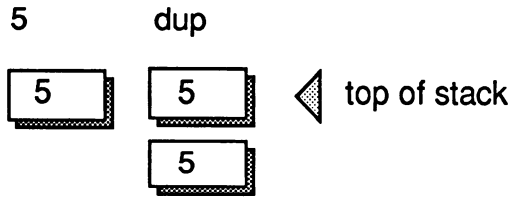
As you can see, the sum is pushed onto the stack after the



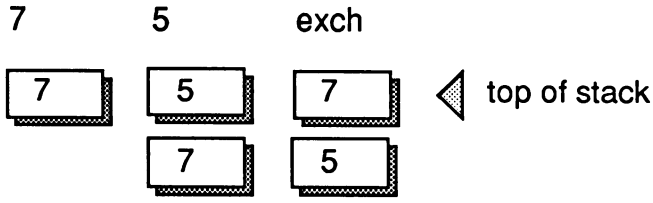
addends are removed. If you're actually trying this code out and want the result to appear on your screen, type = after the code. = pops the result off the top of the stack and echoes it to the output device (it ends up on your screen).

The way that the stack is used may remind you of a RPN calculator. An associate of mine who is studying PostScript prefers to think of the stack in terms of shifting blocks. He's cut a large piece of erasable whiteboard into rectangular sections, and he emulates the stack by writing values onto them and shifting their positions. You may not need to go this far to acquire proficiency with the stack, but you *will* need to retain a mental image of the stack while you're coding. Since many of the simple operators in PostScript don't do much to the stack, it's tempting to look at them simply as functions preceded by their arguments and forget that the stack exists. Unfortunately, this approach won't get you far when you're trying to understand PostScript's stack manipulation operators.

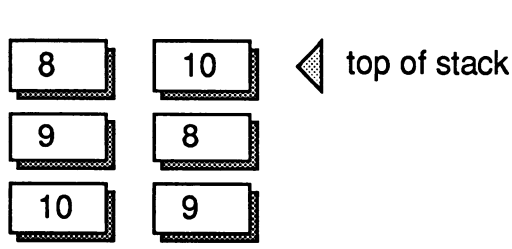
dup duplicates the stack's top element:



exch reverses the order of the top two elements:



roll rotates the order of elements on the stack. It's often used to move the bottommost element to the top of the stack:



roll takes two integer arguments. The first argument determines the number of elements that are affected by the rotation. The second argument specifies how many rotations occur and the direction of these rotations. A negative integer will move elements from the bottom of the stack to the top and a positive integer will move elements from the top of the stack to the bottom. The code necessary to accomplish the above rotation would read 3 -1 roll. Three elements are affected by the operation (if there are any elements beneath the third element, they'll remain in place). The -1 causes the elements to rotate once, moving the third element to the top of the stack. If -2 had been the second argument, 9 would have been moved to the top of the stack first and 8 would have been moved to the top of the stack next.

Obviously, rolls can get quite complex. It's frustratingly similar to Rubik's cube: though it's easy to get a particular element into place, all the other elements get moved around in the process. To help keep track of the stack, most programmers rely on a standard notation that looks like this:

a b c — b c

The letters denote elements on the stack. The ones placed before the dash represent the state of the stack before an operation and those following the dash represent the stack state after the operation. The letter farthest to the right represents the element that's on top of the stack. exch's effect on the stack might be described by a b — b a. For some operations, you may want to use descriptive names instead of letters. The stack effects of add could be written addend1 addend2 — sum. The *PostScript Language Reference Manual*'s reference section describes each PostScript operator using a variant of this notation. add, for example, contains the description num1 num2 **add** sum. The dash has been replaced with the name of the operator. A short dash is used to represent an empty stack, as in - stroke -. Stroke takes nothing from the stack and leaves nothing on the stack after execution.

The stack notation can be used to describe the effect of a single instruction or it can summarize the effect of numerous operations. If I'm writing a tricky program (or debugging code that I had believed to be simple), I'll sometimes use the notation to describe the state of the stack before and after the execution of each line.

Declarations, Types and Dictionaries

Many conventional languages require you to declare a variable's name and type before it can be used. In PostScript, an assignment statement serves as the declaration. The variable's type is determined by the value you assign to it. For instance, / counter 1 def defines an integer variable named "counter" and assigns it the value 1.

In addition to integers, you can assign floating point values (by including decimals), strings (using parentheses as delimiters instead of quotation marks) and arrays (bracketing the elements with '[' and ']'). Arrays can be heterogeneous — the first element might be an integer, the second a floating point value and the third a string. This allows the implementation of complex data structures, though you'll rarely need them.

Let's examine the variable definition a little more closely. The first token, /counter, consists of the variable name counter preceded by a backslash. When PostScript normally encounters a variable name, it pushes the variable's value onto the stack. In this case, however, the backslash indicates that the identifier is to be treated as a name (technically a *literal name*) and pushed onto the stack as such. After placing /counter on the stack, 1 is placed on top of it and the PostScript operator def is invoked. def transforms /counter into a usable variable. First, it checks PostScript's symbol table to see if the named variable already exists. PostScript stores its symbols and their values in a data structure called a dictionary. If def can't find the variable name

in the dictionary, it creates an entry in the dictionary using the supplied name, allocates memory for the data and makes the necessary assignment. A variable can be used in expressions as soon as it's been defined.

The code /counter counter 1 add def increments the counter. counter 1 add pushes counter's value onto the stack and adds one to it, placing the result on the stack. The surrounding code, /counter def is the same as it was in the original declaration. When def is executed, though, it discovers that a variable named counter already exists and simply assigns the new value to it.

def is also used to define procedures. The syntax is the same, except that the body of a procedure is used instead of a variable value. The body consists of PostScript operators (and their arguments) enclosed by curly brackets. When PostScript is interpreting a program and finds that a name refers to a previously defined procedure, the bracketed instructions are executed.

Until now, we've been speaking of a single dictionary for variable and procedure names. PostScript actually uses multiple dictionaries when searching for an identifier. It searches through a list of dictionaries consecutively until the symbol is found or the last dictionary is searched. The order in which the dictionaries are searched is determined by the *dictionary stack*. Normally, the dictionary on the top of this stack is the userdict, and (since it's first in the list) this dictionary holds the variables and procedures created with def. The systemdict dictionary is underneath userdict on the dictionary stack, and it contains the built-in PostScript operators and the system's global variables. There are a few characteristics of PostScript's dictionaries to keep in mind:

- Because of the order of the dictionaries, it's possible to override a built-in operator in systemdict by defining one with an identical name in userdict.
- Symbols in userdict only remain there for the duration of the job. A program cannot use variables or procedures defined during an earlier download.
- Symbols in userdict are global. There's no way of allocating private local variables and then de-allocating them at the end of a procedure.
- There's no way to remove entries from userdict during the execution of a job.
- userdict cannot store an unlimited number of entries.

Efficiency-concious programmers may be tempted to put everything on the stack and move the values into place as needed using the stack operators. This approach will certainly avoid the dictionary overhead, but the code necessary to do this is extremely difficult to write and debug, and the efficiency benefits are often imperceptible. If you don't define variables recklessly, you're not likely to hit userdict's limit (around 200 symbols), and the speed overhead incurred by adding a symbol to the dictionary is minor when compared to the amount of time the LaserWriter takes to place marks on the page.

PostScript in Perspective

To provide a point of reference for programmers accustomed to conventional languages, I'd like to compare the following C function, CenterString, to a similar PostScript function. The C version takes a point, a length and a string pointer as arguments. It draws the string so that it is centered on the interval [point.h, point.h+length] at the height point.v. The PostScript version does much the same thing, drawing on the page instead of the screen. Each line of the PostScript code contains a comment (delimited by a %) that details its effect on the stack.

```
CenterString( StartingPoint, CenterLength, StringPtr )
PointStartingPoint;
intCenterLength;
Str255   *StringPtr; /* A Pascal String */
(
    int Width;
    Width = StringWidth( StringPtr );
    StartingPoint.h = StartingPoint.h +
        (CenterLength - Width)/2;
    MoveTo( StartingPoint.h, StartingPoint.v );
    DrawString( StringPtr );
)
```

Once again, in PostScript:

```
/CenterString % x y clen str -
{
dup           % x y clen str - x y clen str str
stringwidth   % x y clen str str - x y clen str stringx stringy
pop       % x y clen str stringx stringy - x y clen str stringx
3 -1 roll% x y clen str stringx - x y str stringx clen
exch      % x y str stringx clen -  x y str clen stringx
sub           % x y str clen stringx - x y str clen-stringx
2 div         % x y str clen stringx - x y str (clen-stringx)/2
              % offset = clen-stringx/2
4 -1 roll% x y str offset - y str offset x
add           % y str offset x - y str offset+x
3 -1 roll% y str (offset+x) - str (offset+x) y
moveto        % str (offset+x) y - str
show      % str - -
} def
```

Before we examine the PostScript version, let me point out that it's designed to show off some of the stack operators. After you've read through its explanation, I'll present a different version that's less stack-intensive.

Most of the code in CenterString is devoted to arranging the stack so that everything is in order for the mathematical operators. Since we need to reference the string twice in this procedure (once to measure it and once to draw it), dup is used to make a copy of the string. Actually, dup doesn't copy the string; rather, it copies a reference (pointer?) to the string. The next operator, stringwidth, differs somewhat from its QuickDraw relative: the PostScript version returns both a horizontal and a vertical size (just in case you're working with a non-Western font that's written from top to bottom). We don't need the vertical component, so we pop it off the stack. Next, we use roll to move clen to the top of the stack. After an exch, the two elements on the top of the stack are in the correct order for the subtraction. sub leaves the difference between the centering length and the string's width on the top of the stack. To compute the x coordinate of the centering point, we need to halve this difference (2 div) and add the quotient to the x coordinate of the starting point. Since the x coordinate of the starting point is on the bottom of the stack and the stack is 4 elements deep, we use 4 -1 roll to move it to the top

and add the offset to it. Now that the new x coordinate has been determined, we need to move the y coordinate to the top stack, so that the stack will be in order for moveto. The stack is now 3 elements deep and the starting point's y coordinate is on the bottom, so we use 3 -1 roll to bring it to the top of the stack. moveto moves the pen to the x and y coordinates that are on top of the stack, and show draws the string, which is the only remaining element on the stack.

The following version of CenterString avoids most of the stack reorganization described above. On entry, this routine removes the parameters from the stack and places them into global variables:

```
/CenterString     % x y clen str -
{
/str exch def     % x y clen str - x y clen
/clen exch def    % x y clen - x y
/ypos exch def    % x y - x
/xpos exch def    % x - -
clen              %  - - clen
str stringwidth pop    % clen - clen stringx
sub               % clen stringx - clen-stringx
2 div             % clen-stringx - (clen-stringx)/2
xpos add % (clen-stringx)/2 - ((clen-stringx)/2) + xpos
/xpos exch def    % ((clen-stringx)/2) + xpos - -
xpos ypos moveto  % - - -
str show          % - - -
} def
```

exch def is one of the most common sequences in PostScript programs. Recall that def requires a literal name and a value, with the value on top of the stack. To set a variable equal to a value that's on top of the stack, push the literal name on top of the value and then use exch to get the elements in the correct order for def.

In case there's any doubt about how to call CenterString, here's a sample program that uses it:

```
% set the font
/Times-Roman findfont 54 scalefont setfont
% center on 8" page
@ 371 612 (Hello, World!) CenterString
showpage      % draw and eject the page
```

The LaserWriter Driver

The job of the LaserWriter driver is to convert whatever is drawn on the screen (by QuickDraw) into PostScript for the printer. Though the concepts involved in the translation are fairly simple, the implementation is almost frighteningly complex.

When you open a printing port on the Macintosh, the printer driver replaces all of the low-level drawing procedures with procedures that drive the printer. This is done by resetting the thirteen quickdraw primitives that draw basic objects on the screen, with an alternative set of drawing primitives that draw to the printer. These replacement procedures are contained in the printer resource file, *LaserWriter. [For more information on this aspect of the problem, see the printer driver article in the Nov. & Dec. issue of MacTutor. -Ed]* In the case of the LaserWriter driver, the replacement procedures generate a PostScript program that will reproduce the page on the printer. However, the generated code is not "pure" PostScript, it relies heavily on a library of PostScript procedures or macros, that are designed to emulate QuickDraw routines. These procedures are defined in the *LaserPrep* file and are downloaded to the LaserWriter before any pages are printed. You can see what these definitions are by typing cmd-K after hitting the print button in the print dialog.

The *LaserWriter* file contains the replacements for the drawing routines and the PAP (Printing Access Protocol) manager, which coordinates communications with the LaserWriter over AppleTalk. *LaserPrep* contains the definitions of the PostScript routines that emulate QuickDraw. These routines are placed in their own dictionary, and they're downloaded in a special way that allows them to remain in memory until the LaserWriter is turned off. Before the driver prints, it always checks if the LaserPrep dictionary is in memory; if it's not, Laser Prep is downloaded while you watch the message "initializing printer..." Once downloaded, the dictionary consumes at least 62k of LaserWriter memory. Note that Pagemaker uses it's own prep file called AldusPrep. Having both of these in the printer memory at the same time can sometimes run the printer out of memory. Too bad the printer can't automatically swap prep files in and out to avoid this problem!

When I first became acquainted with PostScript, it was clear that many PostScript features weren't accessible through the driver because QuickDraw didn't support them. I considered modifying the driver, but soon found that I lacked the patience necessary to do so. *LaserPrep* contains more than 100 PostScript routines, many of which rely on other routines defined by the driver. Tracing through a single one can involve jumping to fifteen or twenty different places in the source. Even worse, the names given to the routines are literally cryptic (names were reduced to several characters to save space) and there was no LaserPrep documentation. Apple has since documented the LaserPrep routines (Appendix B of the *LaserWriter Reference Manual*), but I still wouldn't recommend driver modifications to anyone lacking endurance. If you're ready to test your patience, you can alter LaserPrep using FEDIT.

There are two common tricks that are indispensible when studying the driver. To save the PostScript code generated by a program that uses the standard LaserWriter driver, press Command-F as the LaserWriter printing dialog disappears. The code will be stored in a TEXT file named "PostScript" instead of being send to the LaserWriter. To save the code generated as well as the driver's PostScript replacement routines, press Command-K as the printing dialog disappears (it will also be stored in a TEXT file named "PostScript").

Under some circumstances, the saved code can simply be downloaded to the LaserWriter. Unfortunately, though, simple downloading is not always enough. Under normal printing conditions, the driver downloads the LaserPrep file (if necessary) as well as any downloadable fonts used in the document. Unless the downloader you're using is intelligent and can perform these functions, you might not be able to get "Command-F" files to print successfully. The downloader presented with this article simply sends PostScript code to the LaserWriter and is best for downloading pure PostScript files. *[It appears that the new version of the driver downloads fonts at the start of the job, because a cmd-K after printing a Reflex Plus report had both the*

prep file and the downloaded laser font definition contained within it, so that it could be directly downloaded to the LaserWriter at a later date. This change may have been made to accomodate the new batch printing of MultiFinder. -Ed]

The Failure of WYSIWYG

In the days of the IBM PC and the daisywheel printer, the choice between a WYSIWYG word processor and a code-based word processor was largely a matter of preference. WYSIWYG was useful because the capabilities of the screen usually matched those of the printer. Though the Macintosh and the LaserWriter represent significant advances in technology, there are differences between what can be done using the Mac's screen with QuickDraw and what can be done on the LaserWriter using PostScript. These differences make WYSIWYG all but useless for dedicated publishing. The production of high-quality printed materials involves typographical refinements (fractional point sizes and leadings, kerning, hairline rules, etc.) much too subtle to be displayed on the Mac's screen. Though there are some page layout programs that provide the user with control over these refinements, they can't be displayed. This forces the user into an iterative process — pages have to be printed and reprinted until everything looks right. Reprinting highlights the flaws in the system software and hardware. The rather baroque translation from QuickDraw to PostScript is slow and the size of the memory-resident LaserPrep dictionary leaves little room on the LaserWriter for downloadable fonts or larger page sizes. Consequently, the printing of documents that use non-resident fonts is slowed to a crawl by the downloading and re-downloading of fonts and AppleTalk gets rather tied up as well. When it takes twenty minutes to print a page, one is lucky to make deadlines — much less worry about refinements.

There are no easy solutions to these problems. The screen resolution difficulties can only be solved by very high resolution displays and QuickDraw modifications. The translation problem can be eased with a large dose of additional memory (impossible on current LaserWriters) and a dedicated font downloader (kludge) . The clean solution, of course, is to provide programs with the ability to generate pure PostScript and output it directly to the LaserWriter. It's clear that this ability will become a common feature of all software (not just page layout programs) in the future; witness the rise of "encapsulated postcript" as a way of blocking out postscript graphics on the page in a program like Pagemaker.

Sources of Information

You'll find the following sources useful when learning PostScript.

PostScript Language Reference Manual
(Red Book)
PostScript Language Tutorial and Cookbook
(Blue Book)
Poscript Language Journal

The Red Book begins with a comprehensive description of PostScript and some important examples. The remainder of the book covers all the PostScript commands and provides some specific information about the LaserWriter. The Blue Book is (as its full title implies) a tutorial with many documented examples. Since these books are published by Addison-Wesley, you can purchase them at most technical bookstores. Watch for the rumored Green book that will cover advanced programming techniques.

Postscript Language Journal is a technical Journal devoted to postscript programming and related Postscript based Mac products. Published by Pipeline Associates, PO Box 5763, Parsippany, NJ 07054. Only $15 per year, quarterly. Highly recommended.

Adobe's telephone support is phenomenal, so phenomenal that it almost makes up for the fact that they distribute no technical notes about PostScript. Call them for help, or information about their PostScript programming classes. *Adobe Systems Incorporated, 1585 Charleston Road, P.O. Box 7900, Mountain View, California 94039-7900 (415) 961-4400.*

Understanding PostScript Programming
by David A. Holzgang.
Sybex, Inc., 1987.

This is the first "3rd Party" book about PostScript and it appears to be a much more readable introduction to the language than Adobe's Blue Book. The example programs aren't quite as complex, which is good, and they're explained in detail. *Sybex, Inc. 2021 Challenger Drive, #100, Alameda, California 94501*

Apple LaserWriter Reference

This document contains invaluable information about printing with the LaserWriter driver. #KNBLRM, from *A.P.D.A., 290 SW 43 Street, Renton, Washington 98055 (206) 251-6548*

"PostScript Halftoning"
by Henry Bortman.

A three part series in the November '86, December '86 and January '87 issues of The Macazine. Bortman gives a very detailed explanation of how shades of gray are rendered by PostScript. *Back Issues/The Macazine, P.O. Box 9802-919, Austin, Texas 78767 (800) 624-2346; $3.75/issue*

Principles of Interactive Computer Graphics
by William M. Newman and Robert F. Sproull.
McGraw-Hill, 1979.

This book doesn't cover PostScript, but it presents some of the mathematical concepts behind PostScript's implementation. It also describes three-dimensional drawing and shading techniques. It's available at technical bookstores.

Hooking Up

There are two ways to run PostScript programs on the LaserWriter. The LaserWriter's PostScript interpreter can operate interactively (like BASIC or FORTH) or it can accept programs downloaded from the Macintosh.

Follow these steps to program interactively:

- Turn the LaserWriter off. On the side with the ports, you'll find a mode switch. Set the switch to 9600.
- Connect the printer to the Macintosh using a null modem cable (the original Apple Personal Modem

cable works). Use the 25-pin (RS-232) serial port on the LaserWriter.

- Start up your favorite communications software on the Macintosh. Setup: 9600 baud, 8 bits, no parity, half duplex.
- Type executive. The PostScript copyright lines will be appear, followed by the system prompt, PS>. You're now in the interactive mode.

The following sections present a downloader that sends PostScript TEXT files to the LaserWriter. There are also a number of text editors that have integral downloaders:

JustText, from Knowledge Engineering, P.O. Box 2139, New York, New York 10116. 212.473.0095.

PostHaste, from Micro Dynamics, Ltd., 8444 16th Street #802, Silver Spring, Maryland 20910. (301) 589-6300.

QuickScript, (under development) from Perimeter, 1608 N. Milwaukee, Chicago, Illinois 60647. (312) 278-9509.

About the Downloader...

The downloader presented here, PS PRINT, provides a convenient way of sending PostScript programs to the LaserWriter. When it launches, you'll be presented with a file dialog containing the names of TEXT files. The file you select will be downloaded to the LaserWriter. When the download ends, you'll be given a chance to save any printer feedback to a text file. Then the original file dialog will appear again, allowing you to download another file. If you click on Cancel, a file dialog containing applications will appear. You can use it to transfer to another application, or return to the Finder by clicking on Cancel.

There are many ways to send PostScript to a printer over AppleTalk. They range from writing your own AppleTalk routines to using high-level print manager functions (see the Apple LaserWriter Reference for details). PS PRINT uses the PAP manager.

The PAP Manager

Communication with the LaserWriter is accomplished via the PAP manager, which provides a high-level interface to AppleTalk. Its code resides in resources that are contained in the LaserWriter file. Oddly, the syntax of the PAP calls doesn't appear to be documented in the *Apple LaserWriter Reference* and there have been modifications to the calls since their initial presentation in Appendix E of *Inside LaserWriter*. The following information about the PAP calls is a revised version of the information that appeared in Appendix E. The declarations are presented in Lightspeed C (note that Lightspeed integers occupy 16 bits). These declarations are here to detail the requirements of the PAP routines. They're not declared this way in the downloader.

```
pascal int PapOpen(ConnectId, LaserName, FlowQuantum,
LaserStatus, OpenState)
int             *ConnectId;
char            *LaserName;
int             FlowQuantum;
papstatusptr    LaserStatus;
int             *OpenState;
```

PapOpen is used to establish a connection between the Macintosh and the LaserWriter over AppleTalk. It is executed asynchronously. This means that, though control returns to the caller, the connection does not necessarily open immediately. If the call returns with a nonzero value, the connection cannot be opened. Otherwise, OpenState will be set to a positive value while the connection is opening. When the connection is open, OpenState will be set to zero. If the opening fails after control has returned to the caller, OpenState will be set to a negative value.

LaserName should point to an AppleTalk EntityName (see the AppleTalk Manager section of *Inside Macintosh*). Usually this will be the name of the printer selected with the Chooser. The Chooser stores this name in the LaserWriter file resource PAPA -8192.

The FlowQuantum is the number of 512 byte buffers that are available for sending and receiving information. The LaserWriter uses a 4096 byte buffer (FlowQuantum = 8), so it's probably most efficient if the Macintosh uses a buffer of that size as well.

The LaserStatus buffer contains a status message that's updated during the opening of the connection. Its structure is declared in the downloader source. The buffer is not cleared by PapOpen, so you should clear it if you intend to use it (PS PRINT does not).

```
pascal int PapWrite(ConnectId, WriteBuffer, WriteSize,
WriteEof, WriteState)
int     ConnectId;
char    *WriteBuffer;
int     WriteSize;
int     WritEof;
int     *WriteState;
```

PapWrite is an asynchronous call that is used to send data to the LaserWriter. Like the other asynchronous PAP calls, it uses the ConnectId that's set by PapOpen and it will return a nonzero value if communications have failed. While it is sending the data, WriteState will be positive. If the transfer is successful, WriteState will be set to 0. Note that this does not mean that the data in the buffer has been processed by the LaserWriter; it has merely been received. If communications fail, WriteState will become negative.

The data being sent is placed in the buffer pointed to by WriteBuffer. The maximum size of this buffer is determined by FlowQuantum (see PapOpen). The application should set WriteSize to the actual size of the data being sent.

Before the last buffer of data is sent to the LaserWriter, the application should set WriteEof to a positive value. This signals the LaserWriter to notify the application (via PapRead) when it has completed processing the job.

```
pascal int PapRead(ConnectId, ReadBuffer, ReadSize, ReadEof,
ReadState)
int     ConnectId;
```

```
char     *ReadBuffer;
int      *DataSize;
int      *ReadEof;
int      *ReadState;
```

PapRead is an asynchronous call that is used to read feedback from the LaserWriter. PAP requires continuous communications between the Macintosh and the LaserWriter. Whenever a connection is open, you must repeatedly call PapRead (using the connection's ConnectId) to ensure that data coming from the LaserWriter is received by the Macintosh.

If communication has failed, PapRead will return a nonzero value, or ReadState will (possibly later) be set to a negative value. If a PapRead has not yet completed, ReadState will be positive. When it's finished, ReadState will be zero.

If there is feedback from the LaserWriter, ReadSize will contain the number of bytes of feedback and the feedback itself will have been placed in the buffer pointed to by ReadBuffer. The size of the buffer is given by FlowQuantum (see PapOpen).

ReadEof is used to monitor the end of the information flow between the Mac and the LaserWriter. Just before the last PapWrite call, the application should set WriteEof to a positive value. This informs the LaserWriter that the final buffer is being sent. Once the buffer has been processed by the LaserWriter, a call to PapRead will return with ReadEof set to a nonzero value. At this point, it's safe to close the connection using PapClose. One can simply close the connection after the final PapWrite's WriteState becomes 0 (when the last buffer has been sent successfully) without waiting for a positive ReadEof, but this will eliminate the possibility of receiving any feedback generated by the final buffer.

```
pascal int PapClose(ConnectId)
int ConnectId;
```

This call closes the connection referenced by ConnectId.

```
pascal int PapUnload()
```

This routine is usually executed after PapClose. It unloads PAP's private data structures and closes any connections that remain open.

```
pascal int PapStatus(LaserName, LaserStatus, LaserNode)
char          *LaserName;
papstatusptr  LaserStatus;
long          *LaserNode;
```

This is a synchronous call that checks the status of the named printer. It can be executed even if there isn't an open connection with the printer.

LaserName should point to an AppleTalk EntityName (see the AppleTalk Manager section of *Inside Macintosh*). Usually this will be the name of the printer selected with the Chooser (see PapOpen above).

LaserStatus is a buffer that will contain the status information requested by the call. Its structure is declared in the downloader source.

The value pointed to by LaserNode should be set to zero the first time the routine is called. PapStatus will set it to the printer's AppleTalk address. Subsequent calls will execute more quickly using the node address.

The PAP manager routines described above (and a few more that aren't relevant to downloaders) are stored as resource PDEF 10 in the LaserWriter file. The entry points for the routines occur at the beginning of the resource. The downloader reads the resource into memory before any of the calls are made. In the downloader, I've created glue routines with the same names as the PAP calls. As mentioned earlier, they do not have the above declarations. In PS PRINT, all of the variables in the above declarations are declared as globals. Each PAP glue routine pushes these globals onto the stack and JSRs to the routine's entry point.

The source for this downloader is based on code presented in Mike Schuster's "Laser Print DA for Postscript" (*MacTutor*, February 1986) and Alan Oppenheimer's downloader, which was published in Appendix E of the original Inside LaserWriter.

Try It

Download the following code to the LaserWriter. Try experimenting with it — I've included some comments regarding parameters that you might want to change.

```
% This routine draws a string 3 times. Each time it is
% offset and drawn with a different shade of gray.
% If you want to experiment, use different values for
% setgray (anything between 0 and 1)

/ShadowShow
{
    /thestring exch def
    currentpoint
    .6 setgray
    thestring show
    moveto
    -4 4 rmoveto
    currentpoint
    .2 setgray
    thestring show
    moveto
    -4 4 rmoveto
    1 setgray
    thestring show
} def

% setscreen is used to change the shading method.
% Normally, shades of gray are rendered with dots
% of various sizes. These arguments to setscreen
% make PostScript render grays with lines of varying
% weights instead of dots. For more info, consult the
% Red Book and Colophon, Volume 2

% the first argument (35) is the # lines/inch and the
% second is the degree at which the lines are drawn.
% Try different values.
35 45 {exch pop abs 1 exch sub} setscreen

% draw and fill the square on which the text will sit
50 280 moveto
50 500 lineto
480 500 lineto
480 280 lineto
50 280 lineto

.9 setgray
fill

% set the font
/Times-Bold findfont 72 scalefont setfont
```


© The Definitive MacTutor, Vol. 4

```
% move into position
100 382 moveto

% skew the space.  concat does a linear
% transformation of the CTM by the matrix
% that precedes it. Read the Red Book and
% dig out your linear algebra textbook.
[ 1 .1 1 1 0 0 ] concat

% Substitute whatever you want for the text wihtin
% the parentheses
(MacTutor) ShadowShow

% inverse-transform these absolute coordinates.
% we want their equivalents in the skewed space
100 318 [ 1 .1 1 1 0 0 ]  itransform moveto
(Magazine) ShadowShow
showpage
```

Downloader Source

```
/*   PS PRINT: A PostScript Downloader
     written by Nicholas Pavkovic
     © Perimeter, 1987.

     PERIMETER
     1608 North Milwaukee Avenue
     Chicago Illinois 60647
     312 . 278 . 9509
     Portions copyrighted by THINK Technologies, Inc.,
     and Apple Computer, Inc.
     Compiled with LightspeedC™
     Libraries linked: MacTraps, sprintf/sscanf*/
#include <MacTypes.h>
#include <QuickDraw.h>
#include <EventMgr.h>
#include <StdFilePkg.h>

/* FileMgr.h Defs (don't need whole file) */
#define fnfErr -43
#define eofErr -39
extern int FSFCBLen : 0x3F6;
extern int BootDrive : 0x210;
extern int SysMap : 0xA58;

/* WindowMgr.h (don't need whole file) */
#define NewWindow      (long) NewWindow
#define dBoxProc 1

/* General */
#define True           1
#define False          0
#define Success        0
#define Failure        -1
#define Fails ==       -1
#define NULL  0L

/* Fall-through functions */

#define IssueRead()    if( issueread() Fails ) return( Failure )
#define CheckIfCancelled()      if( checkifcancelled() Fails ) return( Failure )
#define DisplayStatus()         if( displaystatus() Fails ) return( Failure )
#define OpenLChannel() if( openlchannel()  Fails ) return( Failure)
#define PostMessage(x, y )      if( postmessage(x, y) Fails ) return( Failure )

/* PostMessage selectors */
#define FromPrinter    1
#define FromProgram    0

#define SIMPLEALERT    1000

/* File-related */
int          FErr;
int          FileRef;
SFReply      UserReply;
SFTypeList   Fylz;
Point        SFPoint;
char         *OpenName;
char         NullString;
EventRecord  theEvent;
Handle       DummyHand1;
Rect         DummyRect;
long         DummyType;

/* Windows */
long         PrintWinPtr;
long         DirectWin;

/* Please include InfoWin if you compile and distribute
the program */

long         InfoWin;

char *InfoString = A PostScript Downloader written by
Nicholas Pavkovic\r\r© Perimeter 1987.\rPortions © THINK and
Apple Computer, Inc.\rNon-commercial distribution is permit-
ted.\
    \r\rPERIMETER\r1608 North Milwaukee Avenue\rChicago
Illinois 60647\r312 . 278 . 9509\r \
    Developers of LaserLabels™ and QuickScript™\r\ro Click on
the mouse to continue.;

/* PAP globals  */

typedef struct
{
     long int systemstuff;
     char statusstr[256];
} papstatusrec, *papstatusptr;

papstatusrec LaserStatus;

Handle       pap;
long         papaddr;
int          FlowQuantum;
long unsigned LaserNode;
int          OpenState;
int          WriteState;
int          ReadState;
int          ConnectId;
int          WriteEof;
int          ReadEof;
char         ReadSpace[4096];
char         *WriteBuffer;
int          WriteSize;
int          ReadSize;
char         *LaserName;
Handle       LaserHand;
char         LaserNameBuff[64];
char         VirtualPage[4096];
char         MessageBuff[256];
Handle       FeedbackHandle;
long         FHandleSize;
long         LastStatusDisplay;
/*SimpleAlert:
Implements a general alert used for most of the terminal
error messages.*/
SimpleAlert( m0 )
char *m0;
{
     ParamText( m0, 0L, 0L, 0L );
     NoteAlert( SIMPLEALERT, 0L ) ;
}
```

```
    /*  SFDFilter:
        Filter function for Standard File Dialog.
        Makes the "Open" button context sensitive
        and gives it a heavy outline.*/
    pascal Boolean SFDFilter( dlog, event, itemhit )
    long          dlog;
    EventRecord   *event;
    int           *itemhit;
    {
    if ( event->what == activateEvt && event->modifiers &
activeFlag )
        {
        GetDItem( dlog, getOpen, &DummyType, &DummyHand1,
&DummyRect );
        SetCTitle( DummyHand1, OpenName );
        PenSize(3, 3);
        InsetRect( &DummyRect, -4, -4 );
        FrameRoundRect( &DummyRect, 16, 16);
        PenSize(1,1);
        };
        return( False );
    }
    /*  displaystatus:
        Checks and display status every second,
        even if routine is called more frequently.*/
    displaystatus()
    {
        if( TickCount() - LastStatusDisplay > 60 )
        {
        PapStatus();
        PostMessage( &LaserStatus.statusstr, FromPrinter );
        LastStatusDisplay = TickCount();
        };
        CheckIfCancelled();
    }
    /*  Directions:
        Display text in the directions window.*/
    Directions( text )
    char *text;
    {
        DummyRect.top = 10;
        DummyRect.left = 10;
        DummyRect.right = 342;
        DummyRect.bottom = 80;

        SetPort( DirectWin );
        MoveTo( 10, 16 );
        TextBox( &text[1], (long) text[0], &DummyRect, 0 );
    }
    main()
    {
        char *source, *dest;
        int      counter;
        int       sysVRef;

        InitGraf(&thePort);
        InitFonts();
        InitWindows();
        TEInit();
        InitDialogs(0L);
        FlushEvents(everyEvent, 0);
        InitCursor();
        /*Read PAP routines into memory.
        OpenResFile uses PMSP; we need only to find boot
        drive. See Tech Note #77 for details.        */

        if( FSFCBLen )
        {
            FErr = GetVRefNum( SysMap, &sysVRef );
        }
        else
        {
            sysVRef = BootDrive;
        );

        SetVol( &NullString, sysVRef );

        FErr = OpenResFile(\pLaserWriter);

        if( FErr == -1 )
        {
            SysBeep(1);
            SimpleAlert( \pThe LaserWriter file could not be
found or couldn't be opened. Returning to Finder.);
            return;
        };
        /* Read the PAP code into memory */
        if ( (  pap = GetResource( PDEF, 10 ) ) == NULL ||
ResError() )
        {
            SysBeep(1);
            SimpleAlert( \pThe PAP routines could not be
loaded. Returning to Finder.);
            if( FErr != -1 ) CloseResFile( FErr );
            return;
        };

        /* Detach PAP */
        HLock(pap);
        DetachResource(pap);
        papaddr = (long) *pap;

        /* Read printer name from LaserWriter file */
        if ( (  LaserHand = GetResource( PAPA, -8192 ) ) ==
NULL || ResError() )
        {
            SysBeep(1);
            SimpleAlert( \pThe LaserWriter name could not be
read. Returning to Finder.);
            return;
        };

    /* Create a C string version of the printer name (for
messages) and place it into LaserNameBuff  */

        HLock( LaserHand );
        DetachResource( LaserHand );
        LaserName = *LaserHand;

        /*  Copy the printer name into LaserNameBuff,
        a C string version used for messages to user.*/

        source = (char *) LaserName + 1;
        dest = LaserNameBuff;
        counter = (int) *LaserName;
        while( counter- ) *dest++ = *source++;
        *dest = 0;

        /* Done with LaserWriter file */
        CloseResFile( FErr );

        /* About PS PRINT... */
        DummyRect.left = 62;
        DummyRect.top = 36;
        DummyRect.bottom = 318;
        DummyRect.right = 446;
        InfoWin = NewWindow( 0L, &DummyRect, &NullString,
0xFF, dBoxProc, -1L, False, 0L );
        SetPort( InfoWin );
        TextFace( condense );
        TextFont( 0 );
        MoveTo( 10, 36 );
        DrawString( \pPS PRINT );
        PenPat( gray );
        MoveTo( 64, 26 );
        PenSize( 1, 12 );
        LineTo( 362, 26 );
        PenNormal();
        TextFont( 1 );
```


© The Definitive MacTutor, Vol. 4

```
        DummyRect.top = 46;
        DummyRect.left = 10;
        DummyRect.bottom = 282;
        DummyRect.right = 362;
        TextBox( InfoString, 314L, &DummyRect, -1 );
        while( GetNextEvent( everyEvent, &theEvent) == 0 ||
theEvent.what != mouseDown ) SystemTask();

        DisposeWindow( InfoWin );

        /* Setup instruction window. */
        DummyRect.left = 80;
        DummyRect.top = 36;
        DummyRect.bottom = 90;
        DummyRect.right = 428;

        /*Technically, there's no reason that the directions
window should be a dialog, but doing so avoids the bug
described in Tech Note #99 (SFD doesn't update file list when
new disk inserted).*/

        DirectWin = (long) NewDialog( 0L, &DummyRect,
&NullString, 0xFF, dBoxProc, 0L, 0, 0L, 0L );

        SetPort( DirectWin );
        TextFace( condense );
        TextFont( 0 );

        FeedbackHandle = NewHandle( 0 );

        if( FeedbackHandle == 0 || MemError() )
        {
            SimpleAlert( \pCan't Allocate Feedback Handle.
Returning to Finder. );
            return;
        };

        SFPoint.v = 108;

        /* Download loop */

        while( 1 )
        {
            SetHandleSize( FeedbackHandle, 0L );
            FHandleSize = 0;

            /* Select file to download */

            Directions( \p0 Select a file to download, or\r0
Click on Cancel to transfer or return to the Finder );
            SFPoint.h = 80;
            Fylz[0] = TEXT;
            OpenName = \pDownload;
            SFPGetFile( SFPoint, &NullString,  0L, 1, Fylz,
0L, &UserReply, -4000, SFDFilter );

        /* If file selected, download , otherwise transfer */

            if ( UserReply.good )
            {
                TextDownload( &UserReply.fName,
UserReply.vRefNum );
            }
            else
            {
                Directions( \p0 Select an application to
transfer to, or\r0 Click on Cancel to return to the Finder );
                Fylz[0] = APPL;
                OpenName = \pTransfer;
                SFPGetFile( SFPoint, &NullString,  (ProcPtr)
0, 1, Fylz, 0L, &UserReply, -4000, SFDFilter);

                DisposDialog( DirectWin );
                /* If user cancels, return to Finder */
                if ( !UserReply.good ) return;
                MessageBuff[0] = 0;
                SetVol( &MessageBuff, UserReply.vRefNum );
                Launch( 0, &UserReply.fName );
            };

            /* If feedback received, save it to a file. */
            if( FHandleSize )
            {
                Directions( \p0 Enter a filename to save the
printer feedback, or\r0 Click on Cancel to continue );

                TryAgain:

                SFPoint.h = 104;
                SFPutFile( SFPoint, \pSave the feedback as:,
\pFeedback, 0L, &UserReply );

            /* If  cancelled, loop to downloading code */
                if ( !UserReply.good ) continue;
                FErr = FSOpen( &UserReply.fName,
UserReply.vRefNum, &FileRef );

        /* If the specified file doesn't exist, create it */
                if( FErr == fnfErr )
                {
                SetVol( &NullString, UserReply.vRefNum);
                FErr = Create( &UserReply.fName, 0, EDIT,
TEXT );
                FErr = FSOpen( &UserReply.fName,
UserReply.vRefNum, &FileRef );
                };

                if( FErr != 0 )
                {
                PtoCstr( &UserReply.fName );
                sprintf( &MessageBuff,The file you re-
quested, "%s", is not currently available. Error: %d,
&UserReply.fName, FErr );
                CtoPstr( &MessageBuff );
                SimpleAlert( &MessageBuff );
                goto TryAgain;
                };

                HLock( FeedbackHandle );
                DummyType = FHandleSize;
                SetFPos( FileRef, 1, 0L );
                FSWrite( FileRef, &DummyType, *Feedback-
Handle );
                SetEOF( FileRef, DummyType );
                FSClose( FileRef );
                HUnlock( FeedbackHandle );
            };
        };
    }

    openIchannel()
    {

        long CurrentTime, StartOpening;

        Directions(\p0 Press \021. (Command-Period) to cancel
the download);

        /* Set up print window. */
        DummyRect.top = 108;
        DummyRect.left = 80;
        DummyRect.bottom = 296;
        DummyRect.right = 428;

        PrintWinPtr = NewWindow( 0L, &DummyRect, NullString,
0xFF, dBoxProc, -1L, False, 0L );
        SetPort( PrintWinPtr );
        TextFont( 0 );
```

```
        TextFace( condense );
        PenSize( 1, 1 );
        PenPat( black );

        MoveTo( 10, 24 );
        LineTo( 12, 24 );
        MoveTo( 14, 28 );
        DrawString( \pProgram );
        Move( 2, -4 );
        LineTo( 338, 24 );
        LineTo( 338, 86 );
        LineTo( 10, 86 );
        LineTo( 10, 24 );

        MoveTo( 10, 108 );
        LineTo( 12, 108 );
        MoveTo( 14, 112 );
        DrawString( \pPrinter );
        Move( 2, -4 );
        LineTo( 338, 108 );
        LineTo( 338, 170 );
        LineTo( 10, 170 );
        LineTo( 10, 108 );

        TextFont( 1 );

        /*  Open connection to server   */
        sprintf( &MessageBuff, Looking for printer "%s.",
&LaserNameBuff );
        CtoPstr( &MessageBuff );
        PostMessage( &MessageBuff, FromProgram );
        FlowQuantum = 8;
        ConnectId = 0;
        if ( PapOpen() )
        {
            NoPrinter();
            return( Failure );
        };

        sprintf( &MessageBuff, Establishing connection with
"%s.", &LaserNameBuff );
        CtoPstr( &MessageBuff );
        PostMessage( &MessageBuff, FromPrinter );
        StartOpening = TickCount();
        while( OpenState > 0 )
        {
            DisplayStatus();

            CurrentTime = TickCount();
            if( CurrentTime - StartOpening > 1800 )
            {
                NoPrinter();
                return( Failure );
            };

        CheckIfCancelled();
        };

        if ( OpenState < 0 )
        {
            NoPrinter();
            return( Failure );
        };

    PostMessage( \pConnection established., FromProgram );

        ReadState = 0;
        WriteState = 0;
        ReadEof = 0;
        WriteEof = 0;
        ReadSize = 0;
        WriteSize = 0;

        return( Success );
}

issueread()
{
    long oldsize;
    char *fbptr, *readptr;

    if ( ReadState <= 0 )
    {
/* Negative ReadState indicates communications failure */
        if ( ReadState < 0 )
        {
            EndPCom();
            return( Failure );
        };
        /*
        ReadState == 0 => successful read.
        ReadSize>0 => There's feedback.
        */
        if ( ReadSize > 0 )
        {
            SysBeep(1);

    /*  Feedback messages use ASCII 10
        (linefeed) instead of ASCII 13 (carriage
        return). Convert  them and display.     */

            readptr = &ReadSpace[ ReadSize - 1];
            if( *readptr == 10 ) *readptr = 13;
            *(++readptr) = \0;
            CtoPstr(&ReadSpace);
            PostMessage( &ReadSpace, FromPrinter  );
            Delay( (long int) 150, &DummyHand1);

            /* Update the feedback handle's size and
            copy feedback. */

            oldsize = FHandleSize;
            FHandleSize += ReadSize;
            SetHandleSize( FeedbackHandle, FHandleSize);

            if( MemError() == noErr )
            {
            readptr = &ReadSpace[1];
            fbptr = (char *) *FeedbackHandle + oldsize;
            while( ReadSize- ) *fbptr++ = *readptr++;
            };
        };

        /* Issue another read */
        if (PapRead() )
        {
            EndPCom();
            return( Failure );
        };
        CheckIfCancelled();
        }
        else
        {

    /* Since ReadState > 0, last read not finished  */
            DisplayStatus();
        };
    return( Success );
}

CloseLChannel()
{
    IssueRead();
    /*  set end-of-file  */
    WriteEof = 1;

    /* Send empty buffer */
    WriteBuffer = ;
```


© The Definitive MacTutor, Vol. 4

```
        WriteSize = 0;
        if ( PapWrite() )
        {
            EndPCom();
            return( Failure );
        };
    /* Wait for printer to indicate that processing is over */
        while ( ReadEof == 0) IssueRead();
        PapClose();
        PapUnload();
        DisposeWindow( PrintWinPtr );
        return( Success );
    }
    /*  EndPCom:
        Closes connection immediately.*/

    EndPCom()
    {
        PapClose();
        PapUnload();
        DisposeWindow( PrintWinPtr );
        sprintf( &MessageBuff, Communication with the printer
"%s" has ended., &LaserNameBuff );
        CtoPstr( &MessageBuff );
        SimpleAlert( &MessageBuff );
    }
    /*postmessage:
    Displays messages from printer and program in PrintWin.*/

    postmessage( s, source )
    char *s;
    int      source;
    {
        DummyRect.left = 13;
        DummyRect.right = 335;
        if( source == FromProgram )
        {
            DummyRect.top = 35;
            DummyRect.bottom = 83;
        }
        else
        {
            DummyRect.top = 119;
            DummyRect.bottom = 167;
        };
        TextBox( &s[1], (long) s[0], &DummyRect, 0 );
        CheckIfCancelled();
        return( Success );
    }
    /*  checkifcancelled:
        Checks if user has cancelled the download with
        Command-Period.*/

    checkifcancelled()
    {
        if( GetNextEvent( everyEvent, &theEvent) == 0 )
        {
            SystemTask();
            return( Success );
        };
        if( theEvent.what == keyDown &&
            theEvent.modifiers & cmdKey &&
            (theEvent.message & charCodeMask ) == . )
        {
            EndPCom();
            return( Failure );
        };
    }
    /*  TextDownload:
        Downloads the specified file to the printer*/
        TextDownload( FFileName, Vnum )
    Str255   *FFileName;
    int      Vnum;
    {
        long ItemsRead;
        long FileSize, Remaining;
        Str255   WorkingMessage;

        OpenLChannel();

        FErr = FSOpen( FFileName, Vnum, &FileRef );
        PtoCstr( FFileName );

        if ( FErr != noErr )
        {
            CloseLChannel();
            sprintf( &MessageBuff, Sorry, the file "%s" could
not be opened. Error: %d, FFileName, FErr );
            CtoPstr( &MessageBuff );
            SimpleAlert( &MessageBuff );
            return( Failure );
        };

        GetEOF( FileRef, &FileSize );
        Remaining = FileSize;

        while( Remaining )
        {
            sprintf( &WorkingMessage, Downloading
"%s"\rComplete: %ld%%, FFileName, 100 - ((Remaining * 100 )/
FileSize) );
            CtoPstr( &WorkingMessage );
            PostMessage( &WorkingMessage, FromProgram );

            ItemsRead = 4096;

        FErr = FSRead( FileRef, &ItemsRead, &VirtualPage);

            if ( ItemsRead == 0 && FErr != eofErr )
            {
                CloseLChannel();
                goto ErrorExit;
            }
            else if ( FErr == eofErr )
            {
                WriteSize = (int) ItemsRead;
                if( sendtoprinter( VirtualPage, (int)
ItemsRead ) Fails ) goto ErrorExit;
                FSClose( FileRef );
                sprintf( &MessageBuff, "%s" has been
downloaded., FFileName, FileRef );
                CtoPstr( &MessageBuff );
                PostMessage( &MessageBuff, FromProgram);
                CloseLChannel();
                return( Success );
            };

    if( sendtoprinter( VirtualPage, (int) ItemsRead ) Fails )
        goto ErrorExit;
            Remaining -= ItemsRead;
        };

        ErrorExit:
        FSClose( FileRef );
        sprintf( &MessageBuff, Sorry, the file "%s" could not
be downloaded., FFileName );
        CtoPstr( &MessageBuff );
        SimpleAlert( &MessageBuff );
        return( Failure );
    }
    /*  sendtoprinter:
        General downloading routine (also handles
        buffers > 4096 bytes)*/
    sendtoprinter( bffer, bffsize )
     char    *bffer;
     int     bffsize;
     {
        register char     *bffptr;
```

```
        bffptr = bffer;
        while( bffsize > 0)
        {
            if( bffsize > 4096 )
            {
                WriteSize = 4096;}
            else
            {
                WriteSize = bffsize;};
            WriteBuffer = bffptr;
            IssueRead();
            if ( PapWrite() )
            {
                EndPCom();
                return( Failure );};
            CheckIfCancelled();
        /*  Issue reads to the printer while the
            write is processing.  */
            while ( WriteState > 0  ) IssueRead();
            if ( WriteState < 0 )
            {
                EndPCom();
                return( Failure );};
            bffsize -= 4096;
            bffptr += 4096;
        };
        return( Success );
    }
    /*NoPrinter: Called if communications with printer can't
be established.*/
    NoPrinter()
    {
        PapUnload();
        DisposeWindow( PrintWinPtr );
        sprintf( &MessageBuff, Sorry, the printer "%s" is not
currently available., &LaserNameBuff );
        CtoPstr( &MessageBuff );
        SimpleAlert( &MessageBuff );
        return( Failure );};
    /*PAP glue routines:
    The routines start with a SUB command that makes room for
the result on the stack. Then they push the appropriate PAP
globals (allocated above) onto the stack, and jump to the
routine's entry point, which is at some small offset from the
beginning of pap. When they return from the call, they pop the
result off the stack and put it into retval, which is re-
turned.  */
    PapOpen()
    {
    int retval;
    asm{
    SUBQ.L   #2,A7
    PEA      ConnectId
    MOVE.L   LaserName,-(A7)
    MOVE.W   FlowQuantum,-(A7)
    PEA      LaserStatus
    PEA      OpenState
    MOVE.L   papaddr,A0
    JSR      0(A0)
    MOVE.W   (A7)+,retval}
    return( retval );
    }
    PapRead()
    {
    int retval;
    asm{
    SUBQ.L   #2,A7
    MOVE.W   ConnectId,-(A7)
    PEA      ReadSpace
    PEA      ReadSize
    PEA      ReadEof
    PEA      ReadState
    MOVE.L   papaddr,A0
    JSR      4(A0)
    MOVE.W   (A7)+,retval}
    return( retval );
    }
    PapWrite()
    {
    int retval;
    asm{
    SUBQ.L   #2,A7
    MOVE.W   ConnectId,-(A7)
    MOVE.L   WriteBuffer,-(A7)
    MOVE.W   WriteSize,-(A7)
    MOVE.W   WriteEof,-(A7)
    PEA      WriteState
    MOVE.L   papaddr,A0
    JSR      8(A0)
    MOVE.W   (A7)+,retval}
    return( retval );
    }
    PapStatus()
    {
    int retval;
    asm{
    SUBQ.L   #2,A7
    MOVE.L   LaserName,-(A7)
    PEA      LaserStatus
    PEA      LaserNode
    MOVE.L   papaddr,A0
    JSR      12(A0)
    MOVE.W   (A7)+,retval}
    return( retval );
    }
    PapClose()
    {
        int retval;
        asm{
        SUBQ.L   #2,A7
        MOVE.W   ConnectId,-(A7)
        MOVE.L   papaddr,A0
        JSR      16(A0)
        MOVE.W   (A7)+,retval
        }
        return( retval );
    }
    PapUnload()
    {
        int retval;
        asm{
        SUBQ.L   #2,A7
        MOVE.L   papaddr,A0
        JSR      20(A0)
        MOVE.W   (A7)+,retval      }
        return( retval );
    }
    END OF DOWNLOADER SOURCE
    Resource text for RMaker:
    Type DITL
         ,20963
    2
    *   1
    BtnItem Enabled
    80 62 102 134
    Ok

    *   2
    StatText Enabled
    10 62 65 363
    ^0


    Type ALRT
         ,1000
    78 65 189 446
    20963
    5555
```

C Workshop

Smart Documents

Joel McNamara
Watercourse Software
Bellevue, WA
MacTutor Vol. 4 No. 2

Forked Tongue Programming: Data Forks, Resource Juggling, and Stand Alone Documents

Most developers are quick to realize that a Macintosh file, whether application or not, could easily be an advertisement for Double Mint Gum. Yes, it's two, two, two files in one. As "Inside Macintosh" is quick to point out, "Every file has a resource fork and a data fork (either of which may be empty). The resource fork of an application contains not only the resources used by the application but also the application code itself. The data fork of an application can contain anything an application wants to store there." Big deal. Even most novices know that when an application creates a document, it places the data into the data fork of a file. Pretty basic stuff, even a bit dull and boring for most MacTutor readers.

But let's step beyond the obvious for a moment, and concentrate on the quote "can contain anything an application wants to store there." If we can put anything in there, how about data that only the application could access. No reading in other files, but just operating on the information found within its data fork. Think of it as a stand alone document. Perhaps a word processor that generates a file when double clicked will run itself. Or a chart program that makes files you could modem to a business associate without him needing the creator application. Something a little like "Glue," where you have the ability to view an application's output without needing the application. Runtime versions of data bases incorporate these features. Even Microsoft is rumored to be considering a version of "Excel" that will create stand alone spreadsheets.

Pretty neat stuff, but probably too tricky for the average programmer, right? Wrong. The secret lies within effectively using the resource and data forks.

Stand Alone Document Theory

The reasons for developing such stand alone documents are quite obvious; the ability to disseminate information without the need of a "mother" application to access it; being able to address larger audiences with your output since not everyone needs the creator; not needing sophisticated knowledge of an application to use the information found in a document. All of these points and more make it worthwhile to consider incorporating the ability to generate stand alone documents in your applications.

In thinking about such a project, there are several things to consider. The first is the functionality you want the document to have. More than likely you're not going to want the stand alone file to do everything the original application does. Therefore, you'll need to include some secondary code that will simply display, scroll, print, or do whatever.

But where does one put the code and resources actually needed to run the document? If there is one set of CODE resources that run the main application and another set that will run the stand alone document, won't they conflict and cause problems? Two theories can be used in developing stand alone document generators. The first espouses putting all the required resources into a template file. The generator then simply copies the template, gives the copied file a new name, and inserts the desired information into the data fork. This is the technique used in the stand alone text generator "Take A Letter." The second method is cleaner, eliminating the need for template files. The generator program actually contains the code needed to run a stand alone document, although it may masquerade as another resource. The generator creates a new file, copies in the resources, renames them, fills the data fork, and sets the file's attributes to be an application.

Stand Alone Document Practice

Enough theory, and on to the practice. What will be demonstrated is how to construct a stand alone document generator that works with MacPaint format documents. Three sources are included here. The first (PaintDisplay.c) is code for a program that loads a Paint file stored in its data fork and then shows the upper left hard corner of the bit mapped image. The second listing (PostCard.c) is for an application called PostCard that builds a stand alone Paint document. (The code from PaintDisplay will be inserted into this document generator.) And finally, PostCard.r is the RMaker file. Although the sources are in Lightspeed C, the techniques should be general enough to apply to a different variation of C or even another language. (Lightspeed info: Both projects (PaintDisplay proj and PostCard proj) should include MacTraps plus their own respective source files and be set to build applications. PostCard's creator type should be 'PCRD'.)

The first step is to compile PaintDisplay.c. This source contains a standard algorithm for displaying an image from a MacPaint format file. It initializes everything, opens up its data fork, grabs bytes, puts up a window, displays the left corner of the Paint document, and waits for a mouse click to quit. All in one breath. Don't try to run it quite yet though. If you do, it will be looking in its own data fork for something that isn't there yet, and your curiosity will be rewarded with a bomb.

The only real item in the code that needs discussing, is how one goes about accessing the data fork. To quote from the source:

```
if (GetVol(&thisVolume,&vRef) != noErr)
    ErrorRoutine(FALSE,0);
GetAppParms(&thisProgram,&temp,&thisHandle);
if (FSOpen(thisProgram,vRef,&srcFile) != noErr)
    ErrorRoutine(TRUE,srcFile);
```

The first step is to determine what volume the application is

© The Definitive MacTutor, Vol. 4

on with a GetVol() call. Next, use GetAppParms() to return the name of the current application. Now pass the application's name and the volume reference number to FSOpen(). It will oblige by opening up your application's data fork.

Resource Rascality

Once you've compiled PaintDisplay, the next task is to use RMaker on the PostCard.r file. This file contains the resources for the menus, dialogs, bundles, icons, etc. for the PostCard application. It also has the necessary resources to be used in creating the stand alone document.

An often overlooked feature of RMaker is the ability to copy specific resources from one file to another during the compile stage. By using the ".R" command you could load, let's say, a font from an existing file into the resource file you're currently compiling. The format is:

```
Type FONT = GNRL* the new resource type (FONT)
,8   * the new resource ID
.R   * the RMaker directive
Fonts FONT 12 * the file "Fonts" and the resource
     * (FONT 12) to copy into the new resource
```

RMaker is pretty lax as to letting you get away with things in certain areas. This feature proves beneficial when building program generating applications. As mentioned before, the biggest problem in writing an application that builds another, is where to stick the code that will be spawned off. Obviously you can't have a program with two CODE 0 resources, at least without the potential for serious problems.

The solution is to build a new resource type of your choosing, and then through RMaker, stuff the CODE into it. For example:

```
Type FAKE = GNRL   * creates a resource type called FAKE...
,0                 * with an ID of 0
.R                 * tells RMaker to load in something
SuperEdit CODE 0   * else, i.e., the jump table from
                   * a file called "SuperEdit"
```

This is what's happening in PostCard.r. The previously compiled code from PaintDisplay is being read in, and then assigned a new type and identification. The CODE resource is still the same, but it's gone into hiding by changing its name. It's as easy as that. When the resource file is compiled, you'll have your CODE just sitting there, ready to use, but disguised as something else that won't conflict with the the program that will be using it to create another application.

Taking a peek with ResEdit at the results. FAKE and DUMY are really code resources, but with their new identities, don't conflict with code resources found in the application that will use them.

Two words of advice on using RMaker in such a fashion. The first is, RMaker frowns on using a source file with a space in its name. Therefore, "My File" will cause you problems but "MyFile" will not. The second caveat is to be aware of path names. If all of your formats look all right, but RMaker still is complaining, odds are you have an improperly laid out path. Double check it.

You'll notice in the PostCard.r file, more than just CODE resources are being changed. Remember, all compilers are not

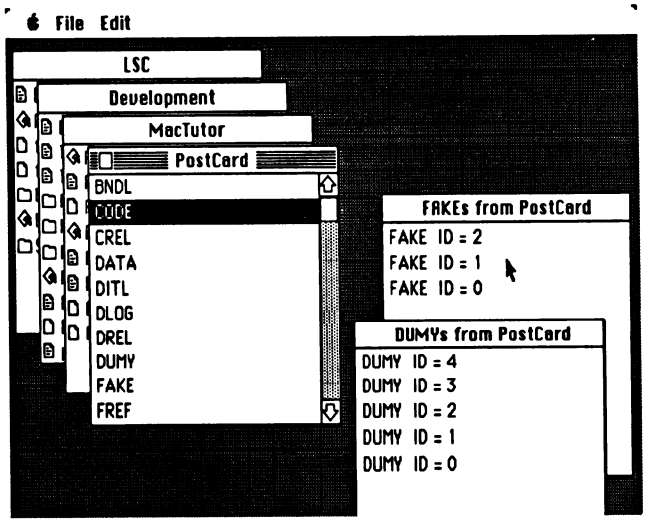


Fig. 1 Using ResEdit to see our fake code resources

created equal. Some will generate just two code segments (for under 32k of code, of course): 0 - the jump table, and 1 - the actual code. Others, Lightspeed C for example, will generate more than two code segments and a variety of other resource types where data is stored. That's what all the CRELs, DATAs, ZEROs, etc. are all about. When you are transferring and converting code resources, be sure you get everything your particular compiler needs for the application to run. At first put everything in, but after a while try some experimenting. At times you'll find you don't need all of the data and support resources.

Resource Moving Without ResEdit

The last step in the process is to compile PostCard.c. The compiled resources from PostCard proj.Rsrc are copied in, and the application is ready to run.

From the Macintosh's perspective, here's what happens when the user selects "Make a PostCard..." from the menu:

The SFPut() routine is called to select a name and location for the new file. CreateResFile() is used to create the named file. Finder attributes are then set, and the resource fork of the new file is opened.

Now the fun stuff. The code determines what the resource reference number of the current application is (in this case PostCard). It then makes use of a programmer defined function called ResTransfer().

```
ResTransfer(sourceRes, sourceID, destRes, destID)
ResType  sourceRes, destRes;
int sourceID, destID;
{
Handle   codeHandle;
     codeHandle = NewHandle(0);
     codeHandle = GetResource(sourceRes,sourceID);
     UseResFile(resRef);
     DetachResource(codeHandle);
     AddResource(codeHandle, destRes, destID, "\p");
     if (ResError() != noErr) {
       DoMessage("\pSorry, couldn't copy the required re-
source.");
          myError = 1;
     }
}
```

ResTransfer() is passed the resource type and ID of the

© The Definitive MacTutor, Vol. 4

resource to be copied, along with what type of resource and what ID number you want the copy to appear as. GetResource() is used to get a handle to the resource to be copied. UseResFile() is then called with the resource reference file number of the application that will be created. This tells the resource manager to only deal with the resource fork of the new stand alone document.

As per "Inside Macintosh," the resource to be copied is detached, and then added to the new application with AddResource(). AddResource() is passed the handle to the resource, along with information concerning type and identification. If for some reason it can't be added, an error handling routine is alerted.

ResTransfer() is called repeatedly for all of the resources needed to be placed in the stand alone document. At first BNDL, FREF, ICN#, etc. are moved in, giving the application its own unique type and icon. Next the FAKE and DUMY resources are copied and then converted back to their original types and IDs.

Once all the resources have been moved and/or changed, UpdateResFile() is used to tell the resource manager the new file needs updating, effectively adding and writing all of the resources. The stand alone document's resource fork is then closed, and UseResFile() is called to revert the current resource fork back to PostCard.

With the resources in place, SFGet() is called to select the Paint format file to be read in. Its data fork is read and then written to the stand alone document's data fork. With no errors reported, the stand alone document has now been created and is ready to run!

Final Thoughts

Needless to say, PostCard ain't very fancy. If one was to do it right, scrolling, desk accessory support, more error handling, perhaps even printing and the ability to save a full MacPaint format document out of it should be included in the PaintDisplay code. Did I just hear the basis for a shareware product?

Also keep in mind that PostCard just demonstrates the idea of stand alone documents. Although application generators like it are useful, stand alone documents would really be best created by say a "WriteNow" or "SuperPaint." Ideally, the entire process would be incorporated as a function in an actual, full blown program where the user would simply select a menu item to create the stand alone file.

One potential argument to the idea of stand alone documents is more code overhead in the original program. However, with a little thought and planning this can easily be avoided by segmenting the program so the main application and the one to be generated would share some common code. Even if an extra 15 or 20k went into the final product, users would likely justify the size increase because of the benefits.

As you can see, creating stand alone documents isn't that difficult. The problems with data and resources can easily be overcome, with the end result being things like self running documentation, double clickable desktop presentations, or stand alone spreadsheets. Hopefully in the near future we'll be seeing more of stand alone documents. But for now, why don't you give your own a try.

```
/*
 * PostCard, version 1.0, Lightspeed C
 * demonstrates using data forks and resource tricks
 * Copyright 1987 by Joel McNamara - All Rights Reserved
 * for MacTutor Magazine
 * initial coding - October 12, 1987
 */
#include "MacTypes.h"
#include "QuickDraw.h"
#include "WindowMgr.h"
#include "EventMgr.h"
#include "MenuMgr.h"
#include "FileMgr.h"
#include "MemoryMgr.h"
#include "ResourceMgr.h"
#include "StdFilePkg.h"
#include "OSUtil.h"
#include "DialogMgr.h"
/* defines... */
#define MessageDLOG    255
#define  AboutDLOG256
#define  HelpDLOG      257
#define  AppleMENU1
#define  About         1
#define  FileMENU      255
#define  Quit          1
#define  CommandMENU   256
#define MakeCard       1
#define Help           3
/* globals... */
Boolean       done;
char          theString[256];
MenuHandle    AppleMenuHndl, FileMenuHndl, CommandMenuHndl;
EventRecord   theEvent;
DialogPtr     theDialog;
int           resRef,dummy,myError;
/* main */
main()
{
    Initialize();
    do{
        MainLoop();
    }while(!done);
}
/* as always... */
Initialize()
{
    InitGraf(&thePort);
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs(0);
    InitCursor();
    FlushEvents(everyEvent, 0);
    SetUpMenus();
    done = FALSE;
    myError = 0;
}
/* make the menus */
SetUpMenus()
{
    AppleMenuHndl = GetMenu(AppleMENU);
    AddResMenu(AppleMenuHndl, 'DRVR');
    InsertMenu(AppleMenuHndl, 0);
    FileMenuHndl = GetMenu(FileMENU);
    InsertMenu(FileMenuHndl, 0);
    CommandMenuHndl = GetMenu(CommandMENU);
    InsertMenu(CommandMenuHndl, 0);
    DrawMenuBar();
```

```
}
/* generic dialog displayer */
DoDialog(theDLOG)
int  theDLOG;
{
    theDialog = GetNewDialog(theDLOG, Ø, -1);
    DoButton(theDialog);
    ModalDialog(Ø, &dummy);
    DisposDialog(theDialog);
}
/* message display routine */
DoMessage(theMessage)
Str255   theMessage;
{
    theDialog = GetNewDialog(MessageDLOG, Ø, -1);
    ParamText(theMessage, "\p", "\p", "\p");
    DoButton(theDialog);
    ModalDialog(Ø, &dummy);
    DisposDialog(theDialog);
}
/* loop until we're done */
MainLoop()
{
    SystemTask();
    if (GetNextEvent(everyEvent,&theEvent)) {
        switch (theEvent.what) {
            case mouseDown:
                DoMouseDown();
                break;
            default:
                break;
        }
    }
}
/* if we have a mouse down */
DoMouseDown()
{
WindowPtrwhichWindow;
int          thePart;
    thePart = FindWindow(theEvent.where,&whichWindow);
    switch(thePart) {
        case inMenuBar:
            DoMenu();
            break;
        case inSysWindow:
            SystemClick(&theEvent, whichWindow);
            break;
        default:
            break;
    }
}
/* if a menu hit then... */
DoMenu()
{
long menuChoice;
    menuChoice = MenuSelect(theEvent.where);
    DoMenuItem(menuChoice);
}
/* which menu was selected? */
DoMenuItem(menuChoice)
long menuChoice;
{
int   theMenu,theItem;
    if (menuChoice != Ø ) {
        theItem = menuChoice;
        theMenu = (menuChoice >> 16);
        switch (theMenu) {
            case AppleMENU:
                DoDA(theItem);
                break;
            case FileMENU:
                done = TRUE;
            break;
            case CommandMENU:
                DoCommands(theItem);
            default:
                break;
        }
        HiliteMenu(Ø);
    }
}
/* the apple menu was hit */
DoDA(theItem)
int theItem;
{
int accNumber;
    if (theItem == About)
        DoDialog(AboutDLOG);
    else{
        GetItem(AppleMenuHndl,theItem,theString);
        accNumber = OpenDeskAcc(theString);
    }
}
/* the commands menu was hit */
DoCommands(theItem)
int theItem;
{
    switch(theItem) {
        case MakeCard:
                ConvertFile();
            break;
        case Help:
                DoDialog(HelpDLOG);
            break;
        default:
            break;
    }
}
/* make a postcard */
ConvertFile()
{
char*        buffer;
long         theEOF;
int          currentResRef,refNum,destRef;
Handle       codeHandle;
Point        myPoint;
SFReply      paintReply, myReply;
SFTypeList   myTypes;
FInfo        myFInfo;
OSErr        myErr;
    myTypes[Ø] = 'PNTG';
    myPoint.h = 90;
    myPoint.v = 90;
    /* first we create the application */
    DoMessage("\pFirst name the PostCard you want to cre-
ate.");
    SFPutFile(myPoint, "\pEnter PostCard name:", "\pMy
PostCard", Ø,
        &myReply);
    if (myReply.good)
    {
        SetVol(Ø, myReply.vRefNum);
        CreateResFile(myReply.fName);
        /* make it what we want... */
        myFInfo.fdType = 'APPL';
        myFInfo.fdCreator = 'PDIS';
        myFInfo.fdFlags = fHasBundle;
        myFInfo.fdLocation.h = Ø;
        myFInfo.fdLocation.v = Ø;
        SetFInfo(myReply.fName, myReply.vRefNum, &myFInfo);
        /* open its resource fork */
        resRef = OpenResFile(myReply.fName);
        if (resRef != -1)
            {
            /* get PostCard's resource ref number and then
            start moving the resources */
                currentResRef = CurResFile();
                /* first the obvious ones */
```


© The Definitive MacTutor, Vol. 4

```
                ResTransfer('BNDL',129, 'BNDL',128);
                ResTransfer('PDIS',0,'PDIS',0);
                ResTransfer('FREF',128,'FREF',128);
                ResTransfer('ICN#',129,'ICN#',128);
                /* now grab the FAKEs and make them CODE */
                ResTransfer('FAKE',0,'CODE',0);
                ResTransfer('FAKE',1,'CODE',1);
                ResTransfer('FAKE',2,'CODE',2);
                /* finally get the DUMYs and convert them */
                ResTransfer('DUMY',0,'CREL',2);
                ResTransfer('DUMY',1,'DATA',0);
                ResTransfer('DUMY',2,'DREL',0);
                ResTransfer('DUMY',3,'STRS',0);
                ResTransfer('DUMY',4,'ZERO',0);
/* update the new application, close it, make PostCard
        the current open resource fork */
                UpdateResFile(resRef);
                CloseResFile(resRef);
                UseResFile(currentResRef);
/* see if we have any errors - open the application */
                if ( (FSOpen(myReply.fName, myReply.vRefNum,
&destRef) != noErr) || (myError == 1) ) {
                    DoMessage("\pSorry a resource error
prevented the PostCard from being created.");
                    FSDelete(myReply.fName,myReply.vRefNum);
                    myError = 0;
                    }
                else
                {
/* here we select the paint file we want to convert */
                DoMessage("\pNow select the Paint file you
want to display.");
                SFGetFile(myPoint, "\p", 0, 1, myTypes, 0,
&paintReply);
                if (paintReply.good)
                {
            /* open, read, and write into the document */
if (FSOpen(paintReply.fName, paintReply.vRefNum, &refNum) ==
noErr)
                    {
                    if ( GetEOF(refNum, &theEOF) != noErr)
                        myError = 1;
                    buffer = NewPtr(theEOF);
            if ( FSRead(refNum, &theEOF, buffer) != noErr)
                    myError = 1;
            if ( FSWrite(destRef, &theEOF, buffer) != noErr)
                    myError = 1;
                    myErr = FSClose(refNum);
                    myErr = FSClose(destRef);
                    if (myError != 1)
                        DoMessage("\pYour PostCard has been
created!");
                    }
                }
                else
                {
                    /* get rid of file we've created */
                    FSDelete(myReply.fName,myReply.vRefNum);
                }
            }
        }
        else
            myError = 1;
    if (myError == 1) {
        /* opps, problems - blitz the file */
        DoMessage("\pSorry an error prevented the PostCard
from being created.");
        FSDelete(myReply.fName,myReply.vRefNum);}
    }
}
/* grab a resource and rename it routine */
ResTransfer(sourceRes, sourceID, destRes, destID)
ResType  sourceRes, destRes;
int      sourceID, destID;
{
Handle   codeHandle;
    codeHandle = NewHandle(0);
        /* get the source resource */
    codeHandle = GetResource(sourceRes,sourceID);
    /* set the current res file to the stand alone document */
    UseResFile(resRef);
    /* detach the source resource... */
    DetachResource(codeHandle);
    /* and add it to the document */
    AddResource(codeHandle, destRes, destID, "\p");
    if (ResError() != noErr) {
        DoMessage("\pSorry, couldn't copy the required
resource.");
        myError = 1;
        }
}
/* fancy button for the modals */
DoButton(myDialog)
DialogPtrmyDialog;
{
short        myType;
Handle       myHandle;
Rect         myBox;
GrafPtr      dummyPort;
    GetPort(&dummyPort);
    SetPort(myDialog);
    GetDItem( myDialog, 1, &myType, &myHandle, &myBox );
    InsetRect(&myBox, -4, -4);
    PenSize(3, 3);
    FrameRoundRect(&myBox, 16, 16);
    PenNormal();
    SetPort(dummyPort);
}
    /*
     * PaintDisplay.c
     * This supplies the code needed for PostCard application
     * to generate a stand alone Paint file.
     * copyright 1987, Joel McNamara - All Rights Reserved
     * for MacTutor Magazine
     * initial coding - October 12, 1987
     */
    #include <MacTypes.h>
    #include <QuickDraw.h>
    #include <WindowMgr.h>
    #include <EventMgr.h>
    #include <MemoryMgr.h>
    #include <SegmentLdr.h>
    /* setup world, display the image, and wait for a click */
    main()
    {   InitGraf(&thePort);
        InitFonts();
        InitWindows();
        InitMenus();
        TEInit();
        InitDialogs(0L);
        InitCursor();
        FlushEvents(everyEvent,0);
        displayPaint();
        while (!Button());
    }
    /* the guts of the paint display program */
    displayPaint()
    {
    char         *srcPtr, *dstPtr, *saveDstPtr, *skipPtr;
    int          srcFile, scanLine, vRef, temp;
    long         srcSize, paintSize;
    Str255       thisProgram;
    StringPtrthisVolume;
    GrafPort     aPort;
    BitMap       theBitMap;
    Rect         myRect;
    WindowPtrmyWind;
    Handle       thisHandle;
```

```
    /* we'll be sloppy with the nonrelocatable pointers and
let them all get cleaned up upon exiting */
        /* the first 512 header bytes to skip */
        srcSize = 512;
        skipPtr = NewPtr(512);
        if (srcPtr == 0L)
           ErrorRoutine(FALSE,0);
        /* the biggest it can be */
        srcPtr = NewPtr(51840);
        if (srcPtr == 0L)
           ErrorRoutine(FALSE,0);
        /* see who and where we are... */
        if (GetVol(&thisVolume,&vRef) != noErr)
            ErrorRoutine(FALSE,0);
        GetAppParms(&thisProgram,&temp,&thisHandle);
        /* then open our data fork and read past the header */
        if (FSOpen(thisProgram,vRef,&srcFile) != noErr)
            ErrorRoutine(TRUE,srcFile);
        if (FSRead(srcFile,&srcSize,skipPtr) != noErr)
           ErrorRoutine(TRUE,srcFile);
        /* see how big we really are... */
        if (GetEOF(srcFile,&paintSize) != noErr)
           ErrorRoutine(TRUE,srcFile);
        paintSize -= 512;
        /* now read the rest of the bytes and close */
        if (FSRead(srcFile,&paintSize,srcPtr) != noErr)
           ErrorRoutine(TRUE,srcFile);
        if (FSClose(srcFile) != noErr)
           ErrorRoutine(TRUE,srcFile);
        /* get a destination pointer... */
        dstPtr = NewPtr(51840);
        if (dstPtr == 0L)
           ErrorRoutine(FALSE,0);
        saveDstPtr = dstPtr;
        scanLine = 1;
        /* and start unpacking the compressed Paint data */
        for (scanLine = 1; scanLine <= 720; scanLine++)
            UnpackBits(&srcPtr,&dstPtr,72);
        /* make a window */
        SetRect(&myRect,12,35,500,325);
        myWind = NewWindow(0L,&myRect,"\p",TRUE,1,-
1L,FALSE,99);
    /* configure our bitmap */
        theBitMap.baseAddr = saveDstPtr;
        theBitMap.rowBytes = 72;
        theBitMap.bounds.top = 0;
        theBitMap.bounds.left = 0;
        theBitMap.bounds.bottom = 72 * 8;
        theBitMap.bounds.right = 720;
        /* and copy the Paint bits into our window */
        CopyBits(&theBitMap, &myWind->portBits, &myWind-
>portRect, &myWind->portRect, srcCopy, 0L);  }
    /* simple no frills, beep and quit */
    ErrorRoutine( closeFile, fileNum )
    Boolean  closeFile;
    int      fileNum;
    {   SysBeep(5);
        if (closeFile)
            FSClose(fileNum);
        ExitToShell();
    }

    * resource file for PostCard.c
    * by Joel McNamara
    * for MacTutor Magazine
    * October 12, 1987
    Development:MacTutor:PostCard proj.Rsrc
    * you'll probably want to change the pathname
    * the bundles, first for PostCard then PaintDisplay
    Type BNDL
    ,128
    PCRD 0
    ICN#
    0 128
    FREF
    0 128

    * this BNDL will be renumbered by the application
    ,129
    PDIS 0
    ICN#
    0 128
    FREF
    0 128

    * we'll use the same FREF for both
    Type FREF
    ,128
    APPL 0

    * PostCard file type
    Type PCRD = STR
    ,0
    PostCard Version 1.0 - 12 Oct 87

    * PaintDISplay file type
    Type PDIS = STR
    ,0
    PaintDisplay Version 1.0 - 12 Oct 87

    * der menu
    Type MENU
    ,1
    \14
    About PostCard...
    (-

    ,255
    File
    Quit

    ,256
    Commands
    Make a PostCard...
    (-
    Help...

    * the dialog items
    Type DITL
    ,255 (4)
    2

    button
    48 272 66 349
    OK

    staticText Disabled
    6 11 43 350
    ^0

    ,256 (4)
    5
    button
    171 86 190 178
    OK

    staticText Disabled
    2 86 20 177
    PostCard 1.0

    staticText Disabled
    24 13 87 263
    PostCard creates an application and then places a Paint
file into the data fork, thus creating a stand alone viewer.

    staticText Disabled
    124 24 158 241
    Copyright 1987, Joel McNamara\0D        All Rights
```

```
Reserved

    staticText Disabled
    100 50 118 208
    for MacTutor Magazine

    ,257 (4)
    4
    button
    166 295 184 406
    OK

    staticText Disabled
    3 4 38 423
    PostCard takes a MacPaint document and turns it into a
stand alone application that displays the upper left corner.

    staticText Disabled
    42 4 91 423
    When the Paint document is selected a new file is created
with CODE resources (FAKE and DUMY - stored within the
PostCard application) copied and renamed into the new file.


    staticText Disabled
    95 4 161 423
    These resources drive the new application by reading its
own data fork (copied in from the Paint file) and then
displaying the bit mapped image.


    * the dialogs
    Type DLOG
    ,255 (4)
    message
    60 68 136 438
    Visible NoGoAway
    1
    0
    255

    ,256 (4)
    about
    52 122 252 394
    Visible NoGoAway
    1
    0
    256

    ,257 (4)
    help
    46 48 242 474
    Visible NoGoAway
    1
    0
    257


    * here's the trickery - we create new resources of type
FAKE, and read
    * the CODE resources from PaintDisplay into them.
    Type FAKE = GNRL
    ,0
    .R
    Development:MacTutor:PaintDisplay CODE 0
    * you'll need to specify a different pathname above

    ,1
    .R
    Development:MacTutor:PaintDisplay CODE 1
    * you'll need to specify a different pathname above

    ,2
    .R
    Development:MacTutor:PaintDisplay CODE 2
    * you'll need to specify a different pathname above


    * now get the rest of the resources we need (as required
by Lightspeed C) -
    * creating dummy resources named DUMY.
    Type DUMY = GNRL
    ,0
    .R
    Development:MacTutor:PaintDisplay CREL 2
    * you'll need to specify a different pathname above

    ,1
    .R
    Development:MacTutor:PaintDisplay DATA 0
    * you'll need to specify a different pathname above

    ,2
    .R
    Development:MacTutor:PaintDisplay DREL 0
    * you'll need to specify a different pathname above

    ,3
    .R
    Development:MacTutor:PaintDisplay STRS 0
    * you'll need to specify a different pathname above

    ,4
    .R
    Development:MacTutor:PaintDisplay ZERO 0
    * you'll need to specify a different pathname above

    * now the application icon
    Type ICN# = GNRL
    ,128
    .H
    00000000 00000000 1FFFF800 20003C00
    20007C00 27FFFC00 2401FC00 2553FC00
    2402F400 2545E400 2409C400 25428400
    24010400 25420400 24003F00 25084080
    24108040 23F13020 2039C814 201E7F8F
    20023007 261F0007 26008007 20006007
    3FFFFFE7 2000021F 1FFFFC07 00000000
    00000000 00000000 00000000 00000000
    * next the mask
    00000000 00000000 1FFFF800 3FFFFC00
    3FFFFC00 3FFFFC00 3FFFFC00 3FFFFC00
    3FFFFC00 3FFFFC00 3FFFFC00 3FFFFC00
    3FFFFC00 3FFFFC00 3FFFFF00 3FFFFF80
    3FFFFFC0 3FFFFFE0 3FFFFFF4 3FFFFFFF
    3FFFFFFF 3FFFFFFF 3FFFFFFF 3FFFFFFF
    3FFFFFFF 3FFFFE1F 1FFFFC07 00000000
    00000000 00000000 00000000 00000000

    * now the stand alone file's icon
    ,129
    00000000 00000000 00000000 00000000
    00000000 00000000 00000000 00000000
    00000000 FFFFFFFF 81D00001 81D000FD
    81F000AD 811000D5 811000AD 811000D5
    811000FD 81100001 80E00001 81F00001
    81F00001 81F80001 81F80001 81E85FFF
    81F80BFF 80D03FFF 80F0FFFF 80703FFF
    8019FFFF 80000001 FFFFFFFF 00000000
    * next the mask
    00000000 00000000 00000000 00000000
    00000000 00000000 00000000 00000000
    00000000 FFFFFFFF FFFFFFFF FFFFFFFF
    FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
    FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
    FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
    FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
    FFFFFFFF FFFFFFFF FFFFFFFF 00000000
```

C Workshop

A Background Task for Measuring Load

Peter Korn
Apple Computer
MacTutor Vol. 4 No. 3

Peter Korn is a full time software quality assurance engineer for Apple Computer, and a full time undergraduate student in Mathematics at UC Berkely, which makes him a busy young man! In this article, he gives us a technique for measuring the activity under multifinder by timing null events in a background task.

True™ Multitasking

I don't have to tell you that multitasking is good. By not having to wait on the computer to finish a task before you can go on to the next one you save time, frustration, and creative energies. There's nothing as frustrating to me as having an idea and wanting to take it somewhere but not being able to do so because my computer is busy printing, or downloading, or compiling (admittedly with LightSpeed C my wait isn't very long). But once we have the concept of multitasking, is multifinder True™ multitasking? Before I tackle that one, I want to give a brief history...

In the beginning there was the Macintosh (128K model). With it I could have a Desk Accessory or three pop up whenever I had an idea that I didn't want to interrupt whatever else I was doing to work on. And, if I hadn't foolishly started a print job or an {up,down} loading session on my modem, I was fine. Some of the usefulness of multitasking already!

When the 512K Mac came along there was more RAM to play with, and some programs took advantage of this RAM to use the Vertical Retrace Manager for more than just a cursor blinker and tick counter. With programs like SuperSpool, I could bring up a DA even while my program was printing. And the DAs were getting bigger too. There were spreadsheets, filers, outliners and word processors and terminal programs (even one called Backdown, that would download for me in the background). And while all of this took even more RAM, the Mac Plus was released shortly and life was pretty ok.

Except that for a program to allow itself to be run 'in the background', it had to go through a series of contortions, and deny itself use of the Memory Manager, and so very few programs were written this way. Certainly Backdown, while nice, wasn't a full-blown terminal emulator. And still there was no way of checking spelling in the background, or compiling, or recalculating a spreadsheet, or what-have-you. The Vertical Retrace Manager just wasn't cut out to be a scheduler.

Even with Switcher I still couldn't do all these sorts of things at once; I could only switch between them when they were idle. And this I had to do by hand (with the exception of ThunderScan, but there's no surprise there, as it shares authors with Switcher).

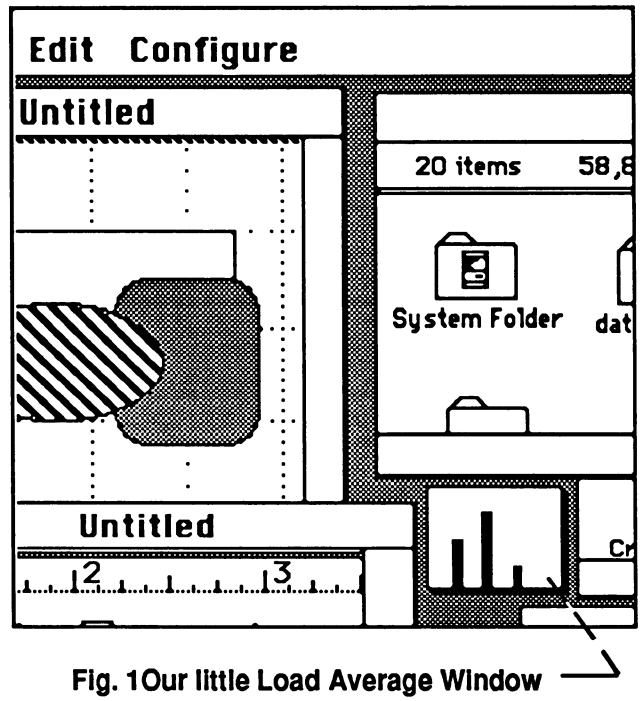


Fig. 1 Our little Load Average Window Measures time between null events

A True™ multitasking system should allow me to run True™ tasks, and have them all going simultaneously (or close to simultaneously—you know what I mean). But an important question is, must this be preemptive? As a user, do I really care how it's all done, so long as my download, spreadsheet recalculation, and mandelbrot plot all continue whilst I'm busy writing a letter to grandma? In fact, if I'm typing my letter, I don't want the computer to be any slower responding to my keystrokes just because it's doing other things—I'm being creative here, and I don't want to have to deal with a sluggish interface that slows *me* down just because I have the computer doing other things as well.

In wider circles, True™ multitasking is considered preemptive multitasking. Each task (program, driver, what-have-you) is assigned some priority number (-10 to 10, say), and some scheduler program arbitrates which task gets how much CPU time when. First it looks at all the programs of the highest priority level, and for all of those that want time (some may not; perhaps their waiting for slow networking i/o, or are finished computing their mandelbrot and haven't been given further instructions), it allocates that time evenly amongst them, interrupting each at some small, regular interval (100 milliseconds, say), and giving time over to the next needy task of equal priority level. When no tasks at that priority level want time, then the scheduler looks at

© The Definitive MacTutor, Vol. 4

tasks of a lower priority level and parcels out time as before to all of them, and so on. If the user doesn't want his creativity disrupted, he merely sets the priority of the task he is working on to something higher than all the other tasks. And in between keystrokes (in the word processor, say), the other tasks will get time to do what they need to do (ie: when the user is idle [for some fraction of a second]).

Apple, in it's infinite wisdom, didn't do it this way. The main reason was for compatibility. Another was because of system architecture (while the Amiga can carry it off without a hardware Memory Management Unit that ensures that whatever memory address a program writes to isn't taken by something else, certain developers [including Apple] have too long thought of the Macintosh as belonging entirely to the program currently running [allowing for those piddling desk accessories, maybe] to pull this off). But is Apple's way also True™ multitasking? Well, if it gets the job done...

Multitasking the Macintosh way

Under the new operating system (System Disk 5.0, which contains System 4.2, Finder 6.0, MultiFinder 1.0, Chooser and Control Panel 3.2, and [aha, so this is where they got the number] LaserWriter Driver 5.0), we can run a program called MultiFinder. MultiFinder, (like Switcher and Servant before them) allows us to load in several programs at once, and switch between them at will, in a very clean fashion (click on window beneath belonging to another program, select the program from the Apple menu, or cycle through the program list on the upper right corner of the MenuBar; this type of program switching is what tech note #180 calls "Major Switching"). Additionally, programs that define themselves as 'backgroundable' can get time when the 'foreground' program (the one who's window is foremost and active) is idle.

This is done by taking control when the foreground program calls GetNextEvent(), EventAvail(), or WaitNextEvent(), a new call under multifinder, and seeing what the foreground program's event is. If the event is not a nullEvent, the foreground program gets it. If the event is a nullEvent, then things get interesting and somewhat confusing.

Null Event Processing

If our foreground program is not MultiFinder friendly, it's doubtless using GetNextEvent(). In this case, MultiFinder doesn't do anything on the first nullEvent. Or the second nullEvent (this is for programs that do work on nullEvents but were written before the program's authors knew about MultiFinder, like MS-Word). But the third nullEvent the foreground program never sees (in the current version of MultiFinder — in later versions GNE() calling apps may only see 1 nullEvent, or perhaps none). Instead MultiFinder looks about and sees if there are any background tasks running (I'll describe those in a moment), and turns control over to one of them (this type of switching is called "Minor Switching" by tech note #180). That background task then calls GNE() or WNE(), and MultiFinder takes over immediately, checks the event, and if it's not a nullEvent, returns control immediately to the foreground program. If it is another nullEvent, MultiFinder checks to see if there are any other background tasks, allocating nullEvents in a round-robin fashion until a non-nullEvent comes along, at which time control is returned to the 'foreground' task, which is returned that non-nullEvent.

For everything to work right, MultiFinder has to make certain assumptions. First, it assumes that *all* programs call GNE(), WNE(), or EA() regularly (which a program is under no obligation to do, though it would be nice...). If a program doesn't, then when it's in the foreground, no background tasks will ever get time. Furthermore, all programs that want to run as background tasks *have* to call one of these three, and call them pretty often, in order to ensure that the foreground task can quickly respond to any user event (a non-nullEvent, in this case). There's no more a way to stop a renegade background task from taking over the CPU the moment it gets it than there is a way to stop a renegade program from making the machine crash — programmers and users and the media will be the judge, not any hardware MMU.

So is it True™ multitasking? You decide.

How to be a (good) background task

Formally all you have to do to be a background task is to have a Size = -1 resource with the *can background* bit set. As a background task, to do any work, you also have to check for, and use, nullEvents. Furthermore, the obedient background task does no more than 50-100 milliseconds of work before calling GNE(), WNE(), or EA() again. for a wonderful example of a disobedient background task, look at Apple's PrintMonitor that's shipping with the 5.0 release (clearly following in MacWrite's lawless ways...). Caveat: in MultiFinder 1.0 there is a bug when using GNE() from the background: you will get no nullEvents when you are first launched—you have to be switched out and then back in again before you start seeing those precious nullEvents.

As a background task, you can do almost anything a foreground task can do. You can write to the screen, read from disk, use the memory manager, play sound, etc. You can even be a foreground task (like a clock that updates the time constantly, and in the foreground allows the user to change from 12 hour to 24 hour display—or like a graphic load average program [which happens to be the subject of this article]). However... you are warned to do all of your work inside of windows. Drawing to the screen is a no-no. And if you are like a certain game I won't mention (but who's initials are Crystal Quest), and you don't draw in a window and expect the user to click the mouse button whilst using your game, you will swiftly find yourself being switched (twitched, as it's sometimes called) out while whatever program's window was underneath the mouse when the mouseDown happened suddenly coming into the foreground. Also you shouldn't put up a Modal Dialog box, as it won't come

up in the foreground but will stay in your layer and the user might never see it. Nor should you do anything that might affect the foreground task, like fiddling with the menuBar or cursor. Lastly, if your program is so uncouth as to change some system parameter (like the mouse tracking ratio, as done by a certain program with the same initials), you should at least be good enough to set it back to what it was when you get twitched out.

Which brings us to the next level of MultiFinder compatibility: accepting *suspend/resume* events (as the bit in the Size = -1 resource is labeled). These programs recognize two new event types that came about from the Switcher days: suspend and resume. A program (any program, not just backgroundable ones) that accepts suspend events doesn't ever get that first nullEvent (on a Major Switch), (unless it's got the background bit set, in which case it will share nullEvents with the other programs that have a background bit set). Instead it will see a suspend event. A suspend event is an app4Evt (theEvent.what = 15) with bit 0 of theEvent.message clear. A resume event is an app4Evt with bit 0 set to 1. A suspend event means that the program is about to be juggled (for some reason the future tense is juggled, and otherwise it's called twitching), and the program should set the state of anything it muddled with back to what it was before it muddled with it and so forth, and then call GNE(), WNE() or EA(). Furthermore, if bit 1 of theEvent.message of the suspend event is set, the application should convert it's internal clipboard to the deskscrap (or vice versa if the event is a resume event) so that other applications can use it. If that bit is clear, then no scrap conversion is necessary.

Another bit in the Size = -1 resource is the *Juggler Aware* bit. If this bit is set, the program uses WNE() instead of GNE() if MultiFinder is running (see the example in my code or in tech note #158 [where I stole it from] for how to do this). WNE() has two additional parameters that it gets passed: sleep (an unsigned long), and mouseRgn (a RgnHandle). Sleep is the number of ticks that the application can go without getting a nullEvent (for things like updating a blinking cursor or clock getting a character from the modem or some such). mouseRgn is a RgnHandle within which the mouse can move and the program needn't be consulted as to it changing shape. An example of this is a terminal emulator, who's cursor is an I-beam when it's over the content region of the window, and an arrow when it's over anything else. NIL cast to the proper variable type can be passed for either of these parameters.

Sayeth Apple: you shouldn't set the *Juggler Aware* bit unless you use WNE(), and recognize suspend/resume events, and are in all ways cognizant of MultiFinder and running with other foreground and background tasks. And what does this involve?

Background tasking programs (in fact, all programs) should now do a number of things in order to better function in the MultiFinder environment. Programs should not do any work on updateEvents (other than updating the window in question with calls to BeginUpdate() and EndUpdate(), if nothing else). If a foreground program's window is moved about, it my reveal a portion of a window underneath it that wasn't revealed before. That program may all of a sudden decide that, on getting it's updateEvent, that it wants to get some more characters from the modem and display them on the screen and such (like another program that I won't mention who's initials [really are] UW). All programs should include a Size = -1 resource, in which, if nothing else, they give a preferred memory partition size that they want to run in. If none is given, 384K is assumed (though this can be edited from the Get Info box in the Finder). Programs that were designed for the 128K Mac (in the 128K Mac days...) wind up taking gobs of RAM for no good reason. All programs need to support Desk Accessories, as it's through this mechanism that Major Switching takes place. Lastly, to better utilize the MultiFinder environment (that allows multiple programs to be autolaunched on startup), programs should save their window positions and states so that the next time that they are invoked the user doesn't have to fuss with moving around all of their windows to where he wants them. The kosher way to do this is with GlobalToLocal() on the upper left and lower right positions of your window (or just the upper left if it's not resizable), and saving that information into the resource or data fork of another file (not the one that the program is in). Then call MoveWindow(savedPoint) to position it before making it visible. Alternately you could save a copy of the window resource into a configuration file. Just keep in mind that reading and writing to the window record itself a no-no (says Apple), 'cause it may change in the future. And by writing to a file other than the program file, you allow yourself to be shared on a network (such as AppleShare or TOPS). Of course, if you don't want to be shared, than this isn't a problem. Unless you are a commercial program with single-user licensing only, you may as well allow yourself to be shared.

Graphic Load Average

When I first saw MultiFinder in action, I immediately thought that some means of measuring the 'load average' of the CPU was needed (after all, most all True™ multitasking systems have this). Being as there was no real way to measure this in the traditional sense (the number of processes the CPU is actively switching through), I decided that instead it would measure the number of nullEvents it got over a period of time, and graph it's results (which is a more meaningful measure under MultiFinder).

Unfortunately someone beat me to it (actually, a lot of people beat me to it, but one sticks out in particular). I have to give credit to Erik Kilk, who posted **Waiting** to the Usenet some 72 hours before I had my program running. Clearly the only way I could salvage my ego was to try to make my program better than his.

Graphic Load Average (GLA) has four load meters, each of which can be set to a different sensitivity and update duration (and width). These settings are set via a hierarchical menu. Additionally it uses WNE(), and will not run if MultiFinder isn't present (a dialog box comes up to nicely inform you of this). Also there is an option to Compress Juggler RAM, which makes a call to the new juggler call MFMaxMem(). While I have yet to see

this call actually do something, certainly having it there somehow improves **GLA** (it also shows how to make the call from LightSpeed C v. 2.13, which doesn't yet support these new calls — see the modified-multifinder.h include file that I made, with the generous help of Ed Tecot of Apple, to use with LightSpeed C). *[Note: The current version of LS C is 2.15. Updates are available from Think, or if purchased from MacTutor, from us. -Ed]*

Graphic Load Average Controls

Each set of triplets controls three things: the width item, which is the width of the displayed graphic bar in the window, the ceiling, which is the number of ticks which will equal a full bar, and the time over which the null events will be averaged. The ceiling parameter is the sensitivity setting, as to how many ticks give a full height bar. The time parameter determines the number of ticks to wait before updating the display. If we set this to five, then we wait five ticks before updating the display with the average number received for that bar. We take the average, and divide it by the ceiling to calculate the height of the bar to be displayed. The ceiling is converted into pixels, divided by the number of ticks that equals a full ceiling to find the number of pixels to display.

By using the four indicators, you can have four different sensitivity settings for how many ticks would result in a full bar display. The maximum setting is a single tick. A tick is a 60th of a second. On each null event, the time is compared, so the average time between null events is being measured. The more bars you display, the longer the time between null events, which means more load on the machine.

Other points of interest and note: putting up a window in a saved position is all very nice, but what if the program is first run on a MacII, and then later is run on a MacPlus. It is important to check that window position is within the bounds of the window (better still, within the dragBounds that you use for MoveWindow()). I cheated in **GLA**, and took the cheap way out when it came to AppleShare. I wrote the configuration information into the same file as I execute from, which means that I cannot be shared under AppleShare (note: even if I was good, and wrote to a separate configuration file, I'd still not automatically be sharable. There's a bit [there's *always* a bit] that I need to set that says "I'm sharable, and won't do anything bad"). *[As we have reported in the past, forcing developers to create hundreds of little configuration files is a poor excuse for Apple's failure to build a resource manager that is sharable. We don't think this is the way to go and don't particulary want our system file filled up with these things. Configuration information should be stored in a document instead. Documents are normally not expected to be sharable at the same time. In this program, Peter does another thing we don't recommend: he opens a specific resource file by file name, a definite no-no. You should obtain the application file name from the finder info and open the file using that name instead of hard-coding a file name into the source. -Ed]*

Dear Phil and Erich
(MultiFinder wish list)

While I'm very happy with all that I can now do with my Macintosh and MultiFinder, there are still things I'd like to see (especially as a programmer).

- I want some way of getting mouse clicks into a window when that mouse click caused a Major Switch (try clicking on GLA's content region in a switch, dragging, and releasing the mouse—GLA's window won't drag)

- I'd like Alert and Dialog boxes that I can bring up via some sort of sublaunch mechanism that puts the dialog/alert 'program' into the foreground and my program in the background so that it can do work while the user takes his/her time responding to the dialog (I'd like a clean mechanism, not have to write my own 'dialog/alert displaying task' especially to do this).

- I want to see more backgroundware from Apple. A MacTerminal that downloads in the background would be very nice. Initializing a floppy diskette in the background would be great (though that requires multi-threaded i/o, another nice feature for a future MultiFinder)

- I'd like some kosher way of notifying a user that I need attention when I'm in the background. What does PrintMonitor know that we lowly developers don't?

- I want to see documentation on this mysterious Layer Manager that's managing all the app layers for me behind my back (and not sending me mouseDowns on Major Switches). In return, I promise to restrain myself from playing with undocumented traps in search of it.

- I'd like language developers to be seeded as early as possible with things like MultiFinder so that I'm not forced to use MPW to program on the cutting edge (try running MPW and MultiFinder and the app your writing on a 2 Meg machine sometime, not to mention the 80% of us who have 1 Meg machines!). Please don't make me write glue for this—that's what the language developers are for. *[A swift kick in the rear to third party develoepers to update their products when they get the new interfaces is also in order. -Ed]*

- I want MultiFinder to somehow guarantee my background task a nullEvent at least every so often, so that my communications software doesn't time out due to it not getting any time. Just a little pre-emption so that connections don't die just because MicroSomeone's software decides to keep the CPU longer than it should have.

- I want the playing of sounds to not kill all user i/o. I want to be able to type characters and switch between programs while sound is playing. How about some more background tasks guys?

```
/* Graphic Load Average
   © 1987 Peter Korn

   This is a little program designed
   to run in the backround under MultiFinder.
   It's purpose:  to graphically indicate
   the 'load' by creating a column chart of
   the load.

*/
/*  Defines  */

#define      NIL               0
#define      nullStopMask      0
#define      MYALERT           128
#define      MYDIALOG          150
#define      MYWINDOW          128
#define      DESK_ID           128
#define      FILE_ID           129
#define      EDIT_ID           130
#define      CONFIGURE_ID      131
#define      BASE_ID           200    /* heirarchical menus */
#define      WINDOW_POSh       150
#define      WINDOW_POSv       151
#define      BAROFFSET         10
#define      WNETrapNum        0x60
#define      UnImplTrapNum     0x9F
#define      SUSPEND_RESUME_EVT   15
#define      SUSPEND_RESUME_BIT   0
#define      CLIPBOARD_BIT     0
#define      MessageDialog     258

/*  Includes */

#include <Quickdraw.h>
#include <MacTypes.h>
#include <FontMgr.h>
#include <WindowMgr.h>
#include <MenuMgr.h>
#include <TextEdit.h>
#include <DialogMgr.h>
#include <EventMgr.h>
#include <DeskMgr.h>
#include <FileMgr.h>
#include <ToolboxUtil.h>
#include <ControlMgr.h>
#include <OSUtil.h>
#include <ResourceMgr.h>
#include <modified_MultiFinder.h>

/*  Globals  */


typedef   struct ConfigurationItem {
   int       value;
   Handle    valueHandle;
   MenuHandle menuHandle;
}  ConfigurationItem;

typedef   struct ConfigurationTriplet {
   ConfigurationItem   ceiling;
   ConfigurationItem   interval;
   ConfigurationItem   width;
};

struct ConfigurationTriplet theConfiguration[4];

long          ticksGoneBy, tickSum[4], tickSumDivider[4];
int           **windowPoshHandle, **windowPosvHandle;
GrafPtr          theCurrentPort;
MenuHandle    deskMenu, fileMenu, editMenu, configureMenu;
WindowPtr     loadWindow;
Str255        daName;
Rect          dragBoundsRect;

/* restartProc  */
restartProc()
{
   quitTime();
}

/* the Switches  */

long   CeilingSwitch (itemNumber)
   int    itemNumber;
{
   long   returnVal;

   switch(itemNumber){
      case 1:
         returnVal = 1;
         break;
      case 2:
         returnVal = 2;
         break;
      case 3:
         returnVal = 3;
         break;
      case 4:
         returnVal = 4;
         break;
      case 5:
         returnVal = 5;
         break;
      case 6:
         returnVal = 6;
         break;
      case 7:
         returnVal = 7;
         break;
      case 8:
         returnVal = 8;
         break;
      case 9:
         returnVal = 9;
         break;
      case 10:
         returnVal = 10;
         break;
      case 11:
         returnVal = 15;
         break;
      case 12:
         returnVal = 20;
         break;
      case 13:
         returnVal = 30;
         break;
      case 14:
         returnVal = 60;
         break;
   }

   return(returnVal);
}

long   IntervalSwitch(itemNumber)
   int itemNumber;
{
   long   returnVal;

   switch(itemNumber){
      case 1:
```


© The Definitive MacTutor, Vol. 4

```
            returnVal = 1;
            break;
        case 2:
            returnVal = 2;
            break;
        case 3:
            returnVal = 5;
            break;
        case 4:
            returnVal = 10;
            break;
        case 5:
            returnVal = 20;
            break;
        case 6:
            returnVal = 60;
            break;
        case 7:
            returnVal = 120;
            break;
        case 8:
            returnVal = 180;
            break;
        case 9:
            returnVal = 600;
            break;
        case 10:
            returnVal = 1200;
            break;
    }

    return(returnVal);
}

long   WidthSwitch(itemNumber)
    int    itemNumber;
{
    long   returnVal;

    switch(itemNumber){
        case 1:
            returnVal = 1;
            break;
        case 2:
            returnVal = 2;
            break;
        case 3:
            returnVal = 3;
            break;
        case 4:
            returnVal = 4;
            break;
        case 5:
            returnVal = 5;
            break;
    }
    return(returnVal);
}

/* doMessage */

/* Politely inform user of messages like no
    multifinder running. Note- this is not
    the kosher way to do things since it doesn't
    involve string resources, which can be edited
    or translated. But it is much faster to develop. */

doMessage (message0,message1,message2,message3)
Str255 message0;
Str255 message1;
Str255 message2;
Str255 message3;
{
DialogPtr dialogP;
int       item;

    ParamText(message0, message1, message2, message3);
    dialogP = GetNewDialog(MessageDialog, (Ptr)NIL, (Ptr)-1);
        if (dialogP == NIL) {
                SysBeep(5);
                ExitToShell();
                }
        else {
                InitCursor();
                ModalDialog((Ptr)NIL, item);
                DisposDialog(dialogP);
            }
}

/* setUpWorld */

setUpWorld()
(
    int       i;
    SysEnvRec theWorld;
    OSErr     err;

    InitGraf(&thePort);
    InitFonts();
    FlushEvents(everyEvent,nullStopMask);
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs(restartProc);

    InitCursor();

    /* check if WNE is implemented, and exit if not
        ...stolen from Tech Note #158            */
    err = SysEnvirons(1, &theWorld);
    if ((theWorld.machineType < 0) ||
        (NGetTrapAddress(WNETrapNum,ToolTrap) ==
NGetTrapAddress(UnImplTrapNum,ToolTrap))) {
            doMessage("\pMultifinder not active","","","");
            ExitToShell();
        }
    SetRect(&dragBoundsRect, screenBits.bounds.left +4,
        screenBits.bounds.top + MBarHeight,
    screenBits.bounds.right -4, screenBits.bounds.bottom -4);

    /* get saved resources & don't let 'em get purged */

    OpenResFile("\pGLA appl");
    /* note: this is not nice. Never hard code a file name. */

    for (i = 0; i < 4; i++) {
        theConfiguration[i].ceiling.valueHandle =
GetResource((ResType) 'INFO', (BASE_ID + i*3));
        HNoPurge(theConfiguration[i].ceiling.menuHandle);
        theConfiguration[i].ceiling.value =
CeilingSwitch(**theConfiguration[i].ceiling.valueHandle);

        theConfiguration[i].interval.valueHandle =
GetResource((ResType) 'INFO', (BASE_ID + 1 + i*3));
        HNoPurge(theConfiguration[i].interval.menuHandle);
        theConfiguration[i].interval.value =
IntervalSwitch(**theConfiguration[i].interval.valueHandle);

        theConfiguration[i].width.valueHandle =
GetResource((ResType) 'INFO', (BASE_ID + 2 + i*3));
        HNoPurge(theConfiguration[i].width.menuHandle);
        theConfiguration[i].width.value =
WidthSwitch(**theConfiguration[i].width.valueHandle);
    }

    /* call ResError(), and set them manually if get error */
```

```
   windowPoshHandle = (int **) GetResource( (ResType) 'INFO',
WINDOW_POSh);
   HNoPurge(windowPoshHandle);

   windowPosvHandle = (int **) GetResource( (ResType) 'INFO',
WINDOW_POSv);
   HNoPurge(windowPosvHandle);

   /* if point not inside dragBoundsRect, set default*/
   if ((**windowPoshHandle < dragBoundsRect.left) ||
      (**windowPoshHandle > dragBoundsRect.right) ||
      (**windowPosvHandle > dragBoundsRect.bottom) ||
      (**windowPosvHandle < dragBoundsRect.top)) {

      **windowPoshHandle = (int) ((dragBoundsRect.right -
dragBoundsRect.left)/2);
      **windowPosvHandle = (int) ((dragBoundsRect.bottom -
dragBoundsRect.top)/2);
   }

   loadWindow = GetNewWindow(MYWINDOW, (Ptr) NIL, (Ptr) -1);
   MoveWindow(loadWindow, **windowPoshHandle, **window-
PosvHandle, FALSE);
   ShowWindow(loadWindow);

   SetPort(loadWindow);
   PenNormal();

}

/* setUpMenus */

setUpMenus()
{
   int    i;

   deskMenu = GetMenu(DESK_ID);
   AddResMenu(deskMenu, 'DRVR');
   InsertMenu(deskMenu, 0);

   fileMenu = GetMenu(FILE_ID);
   InsertMenu(fileMenu, 0);

   editMenu = GetMenu(EDIT_ID);
   InsertMenu(editMenu, 0);

   configureMenu = GetMenu(CONFIGURE_ID);
   InsertMenu(configureMenu, 0);

   for (i = 0; i < 4; i++) {

      theConfiguration[i].ceiling.menuHandle = GetMenu(BASE_ID
+ i*3);
      InsertMenu(theConfiguration[i].ceiling.menuHandle, -1);
      CheckItem(theConfiguration[i].ceiling.menuHandle, (int)
**theConfiguration[i].ceiling.valueHandle, TRUE);

      theConfiguration[i].interval.menuHandle =
GetMenu(BASE_ID + i*3 + 1);
      InsertMenu(theConfiguration[i].interval.menuHandle, -1);
      CheckItem(theConfiguration[i].interval.menuHandle, (int)
**theConfiguration[i].interval.valueHandle, TRUE);

      theConfiguration[i].width.menuHandle = GetMenu(BASE_ID +
i*3 + 2);
      InsertMenu(theConfiguration[i].width.menuHandle, -1);
      CheckItem(theConfiguration[i].width.menuHandle, (int)
**theConfiguration[i].width.valueHandle, TRUE);
   }

   DrawMenuBar();
}
```

```
/* doMenu */

doMenu(menuResult)
   long   menuResult;
{
   int       menuID, itemNumber, newHPos;
   int       newVPos, itemHit, i, configSetting;
   long      longPointer;
   ProcPtr   myFilterProc;

/* if (menuResult == (long) NIL)
      menuResult = MenuChoice(); */

   menuID    = HiWord(menuResult);
   itemNumber= LoWord(menuResult);

   switch(menuID)
      {
      case DESK_ID:       /* Apple Menu    */
         if (itemNumber == 1){
            Alert(MYALERT, NIL);
            break;
         }
         else if (itemNumber == 0)  /* b/c of nested menus */
            break;
         else
         {
            GetItem(deskMenu, itemNumber, &daName);
            GetPort(&theCurrentPort);
            OpenDeskAcc(&daName);
            SetPort(theCurrentPort);

            break;
         }
         break;

      case FILE_ID:          /* the File Menu */
         switch(itemNumber)
            {
            case 1:          /* the print command   */
               break;
            case 2:          /* the quit command */
               quitTime();
               break;
         }
         break;

      case EDIT_ID:       /* the Edit Menu */
         switch(itemNumber){
            case 1:          /* the undo command */
               break;
            case 2:          /* just dashed lines   */
               break;
            case 3:          /* the cut command  */
               break;
            case 4:          /* the copy command */
               break;
            case 5:          /* the paste command   */
               break;
            case 6:          /* the clear command   */
               break;
         }
         break;

      case CONFIGURE_ID:      /* the Configure Menu  */
         switch(itemNumber){
            case 1:       /* Set Time: Heirarchical */
            case 2:    /* Average Over: Heirarchical */
            case 3:       /* Set Width:Heirarchical */
               break;
            case 4:       /* spacer */
               break;
```

```
            case 5:   /* Set Time:     Heirarchical */
            case 6:   /* Average Over: Heirarchical */
            case 7:   /* Set Width:    Heirarchical */
               break;
            case 8:       /* spacer */
               break;
            case 9:   /* Set Time:     Heirarchical */
            case 10:  /* Average Over: Heirarchical */
            case 11:  /* Set Width:    Heirarchical */
               break;
            case 12:  /* spacer */
               break;
            case 13:  /* Set Time:     Heirarchical */
            case 14:  /* Average Over: Heirarchical */
            case 15:  /* Set Width:    Heirarchical */
               break;
            case 16:  /* spacer */
               break;
            case 17:  /* compress juggler RAM... */
               MFMaxMem(&longPointer);
               break;
         } /* end switch */
         break;

      case BASE_ID:
      case (BASE_ID + 3):
      case (BASE_ID + 6):
      case (BASE_ID + 9):

         configSetting = ((menuID - BASE_ID)/3);
         CheckItem(theConfiguration[configSetting].ceiling.menuHandle,
(int) **theConfiguration[configSetting].ceiling.valueHandle,
FALSE);
         **theConfiguration[configSetting].ceiling.valueHandle
= (SignedByte) itemNumber;
         CheckItem(theConfiguration[configSetting].ceiling.menuHandle,
(int) **theConfiguration[configSetting].ceiling.valueHandle,
TRUE);
         theConfiguration[configSetting].ceiling.value =
CeilingSwitch(itemNumber);
         break;

      case (BASE_ID + 1):
      case (BASE_ID + 4):
      case (BASE_ID + 7):
      case (BASE_ID + 10):
         configSetting = (menuID - BASE_ID - 1)/3;
         CheckItem(theConfiguration[configSetting].interval.menuHandle,
(int) **theConfiguration[configSetting].interval.valueHandle,
FALSE);
         **theConfiguration[configSetting].interval.valueHandle
= (SignedByte) (itemNumber);
         CheckItem(theConfiguration[configSetting].interval.menuHandle,
(int) **theConfiguration[configSetting].interval.valueHandle,
TRUE);
         theConfiguration[configSetting].interval.value =
IntervalSwitch(itemNumber);
         break;

      case (BASE_ID + 2):
      case (BASE_ID + 5):
      case (BASE_ID + 8):
      case (BASE_ID + 11):
         configSetting = (menuID - BASE_ID - 2)/3;
         CheckItem(theConfiguration[configSetting].width.menuHandle,
(int) **theConfiguration[configSetting].width.valueHandle,
FALSE);
         **theConfiguration[configSetting].width.valueHandle
= (SignedByte) itemNumber;
         CheckItem(theConfiguration[configSetting].width.menuHandle,
(int) **theConfiguration[configSetting].width.valueHandle,
TRUE);
         theConfiguration[configSetting].width.value =
WidthSwitch(itemNumber);
         break;

   }
   HiliteMenu(0);
}

/* doLoad */

doLoad()
{
   long      currentTicks, deltaTicks, barHeight, barWidth;
   long      leftEdge, rightEdge, topEdge, bottomEdge;
   Rect      loadBar, eraseBar;
   int       i;


   /* sample the load  */
   currentTicks = TickCount();
   deltaTicks = currentTicks - ticksGoneBy;
   ticksGoneBy = currentTicks;

   /* increment accumulators and draw load */
   for (i = 0; i < 4; i++) {
      tickSum[i] += deltaTicks;
      tickSumDivider[i]++;
      if (tickSum[i] > theConfiguration[i].interval.value) {

         leftEdge = 5 + (i * BAROFFSET);
         bottomEdge = 35; /* will become resizable  */
         topEdge = 0;
         barWidth = theConfiguration[i].width.value;

         barHeight = (40 - ((tickSum[i] * 35)/
(tickSumDivider[i] * theConfiguration[i].ceiling.value)));
   /* eliminate decimals */
         SetRect (&loadBar, leftEdge, barHeight, leftEdge +
barWidth, bottomEdge);
         SetRect (&eraseBar, leftEdge, topEdge, leftEdge +
barWidth, barHeight);
         EraseRect(&eraseBar);
         PaintRect(&loadBar);

         tickSum[i] = 0;
         tickSumDivider[i] = 0;
      }  /* endif  */
   }  /* end for   */
}

/* quitTime */

quitTime()
{           /* time to write resources, etc.     */
   int    i;
   Point  p;

   for (i = 0; i < 4; i++) {
      ChangedResource(theConfiguration[i].ceiling.valueHandle);
      ChangedResource(theConfiguration[i].interval.valueHandle);
      ChangedResource(theConfiguration[i].width.valueHandle);
   }

   SetPort(loadWindow);

   p.h = 0;
   p.v = 0;

   LocalToGlobal(&p);

   **windowPoshHandle = p.h;
   **windowPosvHandle = p.v;

   ChangedResource((Handle) windowPosvHandle);
```

```
    ChangedResource((Handle) windowPoshHandle);

    ExitToShell();
}


/* main */

main()
{

    /* variables */

    EventRecord   theEvent;
    Point         eventPt;
    char          eventChar;
    WindowPtr     whichWindow, eventWindow;
    short  windowCode, stillInGoAway, controlCode, whichItem,
                  scrapIndex;

    setUpWorld();
    setUpMenus();


    while (TRUE) {
       WaitNextEvent(everyEvent, &theEvent, (long) NIL,
(RgnHandle) NIL);
       switch (theEvent.what){
          case SUSPEND_RESUME_EVT:
             if ( ! BitTst( (Ptr) theEvent.message,
SUSPEND_RESUME_BIT))    /* suspend event */
                ;
             else (BitTst( (Ptr) theEvent.message,
SUSPEND_RESUME_BIT))    /* resume event  */
                ;
             if ( ! BitTst( (Ptr) theEvent.message,
CLIPBOARD_BIT))      /* don't convert clipboard */
                ;
             else (BitTst( (Ptr) theEvent.message,
CLIPBOARD_BIT))      /* convert clipboard    */
                ;
             break;
          case mouseDown:
             eventPt = theEvent.where;
             windowCode = FindWindow(eventPt, &eventWindow);
             switch (windowCode) {
                case inSysWindow:
                   SystemClick(&theEvent, eventWindow);
                   break;
                case inContent:
                   /* won't work on minor switch
                      b/c of compatibility issues w/Excel */
                case inGrow:
                case inGoAway:
                   if (eventWindow == loadWindow)
                      DragWindow(eventWindow, eventPt,
&dragBoundsRect);
                   break;
                case inDrag:
                   DragWindow(eventWindow, eventPt,
&dragBoundsRect);
                   break;
                case inDesk:
                      break;
                case inMenuBar:
                   doMenu(MenuSelect(theEvent.where));
                   break;
                default:
                   break;
             }  /* end switch(windowCode) */
          break;

          case keyDown:
          case autoKey:
             eventChar = theEvent.message;
             if (theEvent.modifiers & cmdKey) {
                doMenu(MenuKey(eventChar));
                HiliteMenu(0);
             }
             break;
          case activateEvt:
             break;
          case updateEvt:  /* absolutely necessary*/
             BeginUpdate(theEvent.message);
             EndUpdate(theEvent.message);
             break;
          case nullEvent:
             doLoad();
             break;
          default: ;
       } /* end of case theEvent.what */
    } /* while */
}
```

```
/*
   MultiFinder.h - MultiFinder Interfaces

   C Interface to the Macintosh Libraries
   Copyright Apple Computer,Inc.  1987
   All rights reserved.
*/

#ifndef __MULTIFIN__
#define __MULTIFIN__
#ifndef __EVENTS__
#endif
#ifndef __QUICKDRAW__
#endif


/* MultiFinder dispatch trap */

#define _OSDISPATCH        0xA88F
#define   _WAITNEXTEVENT          0xA860



/* Routine selector values */

#define mfMaxMemSel           21
#define mfFreeMemSel          24
#define mfTempNewHandleSel        29
#define mfTempHLockSel            30
#define mfTempHUnLockSel          31
#define mfTempDisposHandleSel     32


#define MFMaxMem(grow) \
   cMFMaxMem(grow,mfMaxMemSel)

#define MFFreeMem() \
   cMFFreeMem(mfFreeMemSel)

#define MFTempNewHandle(logicalSize,resultCode) \
   cMFTempNewHandle(logicalSize,resultCode,mfTempNewHandleSel)

#define MFTempHLock(h,resultCode) \
   cMFTempHLock(h,resultCode,mfTempHLockSel)

#define MFTempHUnLock(h,resultCode) \
   cMFTempHUnLock(h,resultCode,mfTempHUnLockSel)

#define MFTempDisposHandle(h,resultCode) \
   cMFTempDisposHandle(h,resultCode,mfTempDisposHandleSel)
```

```
/* WaitNextEvent was released in MPW 2.0 interfaces. It is
   commented out below to avoid a duplicate declaration.
   Uncomment only if working with earlier interfaces. */

pascal Boolean WaitNextEvent(mask,event,sleep,mouseRgn)
   unsigned short mask;
   EventRecord *event;
   unsigned long sleep;
   RgnHandle mouseRgn;
   {
      asm {
         SUBQ.L #2, SP
         MOVE.W 20(A6), -(SP)    /* short mask*/
         MOVE.L 16(A6), -(SP)    /* long *event  */
         MOVE.L 12(A6), -(SP)    /* long sleep*/
         MOVE.L 8(A6), -(SP)     /* long handle  */
         DC.W   _WAITNEXTEVENT
         MOVE.W (SP)+, 22(A6)    /* return stack pointer
state*/
      }
   }

pascal long cMFMaxMem(grow, SW)
   long         *grow;
   short        SW;
   {
      asm(
         SUBQ.L #4, SP
         MOVE.L 10(A6), -(SP)    /* long *grow*/
         MOVE.W 8(A6), -(SP)     /* short SW  */
         DC.W   _OSDISPATCH
         MOVE.L (SP)+, 14(A6)    /* return stack pointer
state */
      }
   }

pascal Size cMFFreeMem(SW)
   short        SW;
   {
      asm(
         DC.W   _OSDISPATCH
      )
   }

pascal Handle cMFTempNewHandle(logicalSize, resultCode, SW)
   unsigned long logicalSize;
   short        *resultCode;
   short        SW;
   {
      asm(
         DC.W   _OSDISPATCH
      )
   }

pascal void cMFTempHLock(h, resultCode, SW)
   Handle       h;
   short        *resultCode;
   short        SW;
   {
      asm(
         DC.W   _OSDISPATCH
      )
   }

pascal void cMFTempHUnLock(h, resultCode, SW)
   Handle       h;
   short        *resultCode;
   short        SW;
   {
      asm(
         DC.W   _OSDISPATCH
      )
   }

pascal void cMFTempDisposHandle(h, resultCode, SW)
   Handle       h;
   short        *resultCode;
   short        SW;
   {
      asm(
         DC.W   _OSDISPATCH
      )
   }


#endif
```

```
*Graphic Load Appl.R
*
GLA.rsrc
RSRCWAKA

Type WAKA = STR
   ,0
Graphic Load Average \0D© by Peter Korn \0Dver 28 JAN 1988

Type FREF
,128
APPL 0

Type BNDL
,128
WAKA 0
ICN#
0 128
FREF
0 128

TYPE WIND
BarWindow,128
load
40 350 80 400
Invisible NoGoAway
3
0

TYPE SIZE = GNRL
JugglerSize, -1
5800 0000 8000 0000 8000   ;; gotten by looking at Size = -1
resource as general

TYPE MENU
AppleMenu, 128
\14
About Load Average
(-

FileMenu, 129
File
(Print/P
Quit/Q

EditMenu, 130
Edit
(Undo/Z
(-
(Cut/X
(Copy/C
(Paste/P
(Clear/B

ConfigureMenu, 131
Configure
Tick ceiling/\1B!\C8
```

```
Interval time/\1B!\C9
Measurement width/\1B!\CA
(-
Tick ceiling/\1B!\CB
Interval time/\1B!\CC
Measurement width/\1B!\CD
(-
Tick ceiling/\1B!\CE
Interval time/\1B!\CF
Measurement width/\1B!\D0
(-
Tick ceiling/\1B!\D1
Interval time/\1B!\D2
Measurement width/\1B!\D3
(-
Compress juggler RAM

Ceiling1Menu, 200
Ceiling1
1
2
3
4
5
6
7
8
9
10
15
20
30
60

Interval1Menu, 201
Interval1
1
2
5
10
20
60
120
180
600
1200

Width1Menu, 202
Width1
1
2
3
4
5

Ceiling2Menu, 203
Ceiling2
1
2
3
4
5
6
7
8
9
10
15
20
30
60

Interval2Menu, 204
Interval2
1
2
5
10
20
60
120
180
600
1200

Width2Menu, 205
Width2
1
2
3
4
5

Ceiling3Menu, 206
Ceiling3
1
2
3
4
5
6
7
8
9
10
15
20
30
60

Interval3Menu, 207
Interval3
1
2
5
10
20
60
120
180
600
1200

Width3Menu, 208
Width3
1
2
3
4
5

Ceiling4Menu, 209
Ceiling4
1
2
3
4
5
6
7
8
9
10
15
20
30
60

Interval4Menu, 210
Interval4
1
2
5
10
20
60
120
180
600
1200

Width4Menu, 211
Width4
1
2
3
4
5

* Program Messages Dialog box...
type DLOG
    ,258
Program Messages
100 100 200 400
Visible NoGoAway
1
0
258

type DITL
    ,258
3
BtnItem Enabled
65 230 95 285
OK

StatText Disabled
15 60 85 222
^0\0D^1\0D^2\0D^3

IconItem Disabled
10 10 42 42
1

TYPE INFO = GNRL
Window Pos h, 150
0060        ;; initial hor window position

Window Pos v, 151
0060        ;; initial ver window position

Ceiling1, 200
01          ;; initial Ceiling1 value

Interval1, 201
01          ;; initial Interval1 value

Width1, 202
05          ;; initial Width1 value

Ceiling2, 203
02          ;; initial Ceiling2 value

Interval2, 204
02          ;; initial Interval2 value

Width2, 205
05          ;; initial Width2 value
```


© The Definitive MacTutor, Vol. 4

```
Ceiling3, 206
03              ;; initial Ceiling3 value

Interval3, 207
03              ;; initial Interval3 value

Width3, 208
05              ;; initial Width3 value

Ceiling4, 209
04              ;; initial Ceiling4 value

Interval4, 210
05              ;; initial Interval4 value

Width4, 211
05              ;; initial Width4 value


TYPE ALRT
MyAlert, 128 (2)
79 139 192 450
128
4444

TYPE DITL
MyDialogList, 128 (2)
4
   ButtonItem Enabled
   81 127 101 177
   OK

   StatText Enabled
   8 78 28 228
   Graphic Load Average

   StatText Enabled
   28 78 48 228
   written by Peter Korn

   StatText Enabled
   53 9 72 294
   Portions of this code © Think Technologies

Type ICN# = GNRL
  ,128 (0)
.H
0000 0000 0000 0000 00FF FF80 0300 0060
0400 0010 0800 0008 1000 0004 1000 0004
2000 0002 2000 0002 2000 0002 2000 F002
2000 F002 2000 F002 2000 F002 2000 F002
21E0 F002 21E0 F002 21E0 F002 21E0 F0E2
21E0 F0E2 21E0 F0E2 21E0 F0E2 21E0 F0E2
11E0 F0E4 11E0 F0E4 09E0 F0E8 05E0 F0F0
03E0 F0E0 00FF FF80 0000 0000 0000 0000
*
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
```

C Workshop

Tear-off Menus & Floating Palettes

Don Melton and Mike Ritter
Impulse Technologies, Inc.
Santa Clara, California
MacTutor Vol. 4 No. 4

Don Melton is the Computer Graphics Specialist for the San Jose Mercury News and does technical support for the Knight-Ridder Graphics Network. His partner, Mike Ritter, is a Research Scientist at Lockheed's Palo Alto Research Laboratories in the Electro-optics Department. In their spare time they do MacHacking for their own company, Impulse Technologies, Inc.

Introduction

Why TearOffPalette? Well, ever since we saw HyperCard we've been wondering, just like everyone else, how Bill Atkinson did that impressive tear-off menu effect. Of course, the wizardry Andy Hertzfield performed with Radius's new monitors caught our eye, too. If you haven't seen Andy's ROM-defying performance, take a look — he makes *any* menu tear off!

And now everyone wants to get into the act.

Daryl Lovato gave many of us our first clues toward making menus leave their home in his article titled "Tear-Off Menus Explained" in December's MacTutor. But he still left many questions unanswered, like how do you get the darn palettes to 'float' above other application windows (which we call documents to differentiate them from the floating palettes). And he didn't use those stylish HyperCard title bars in his palettes either.

We also noticed that all of the applications we saw which used tear-off menus had some rather strange 'features' in them. Our first attempt to do the tear-off trick simply mimicked HyperCard. The constant flashing of menu items soon drove us batty. Then there's Bill's highlighting (or lack thereof) of his main window when a menu is torn off.

Then we saw the new MacPaint. Sure, the highlighting is better but the palettes are in a fixed order! Even HyperCard doesn't do that. Two steps forward and one step back. That's progress.

If only Apple would just torture Andy Hertzfield until he lets them release his code as system software. Alas, Radius has to maintain its market advantage. Maybe if everybody buys enough Radius monitors Burrell Smith may let the cat out of the bag.

Until then ... we decided to show you how it can be done. And how to make it nice and modular too — with one application, an MDEF and a WDEF. Using our code any developer can implement tear-off menus and floating palettes in their application. Really. As a bonus, our source code will also demonstrate how to:

- Work with MultiFinder using WaitNextEvent()
- Use Color QuickDraw to implement a full-color palette
- Respond to suspend/resume events
- Write a fully-working, albeit simple, grow zone function

And it has extensive error checking built right in. Enough to function correctly with strict trap discipline and heap check, scramble and purge, all set in TMON, while it runs concurrently under MultiFinder with ResEdit and any Microsoft application. Just about the nastiest environment a Mac developer could imagine.

It also (*ta da!*) works with any Macintosh. That's right, even those lonely ones in the corner with 64K ROMS.

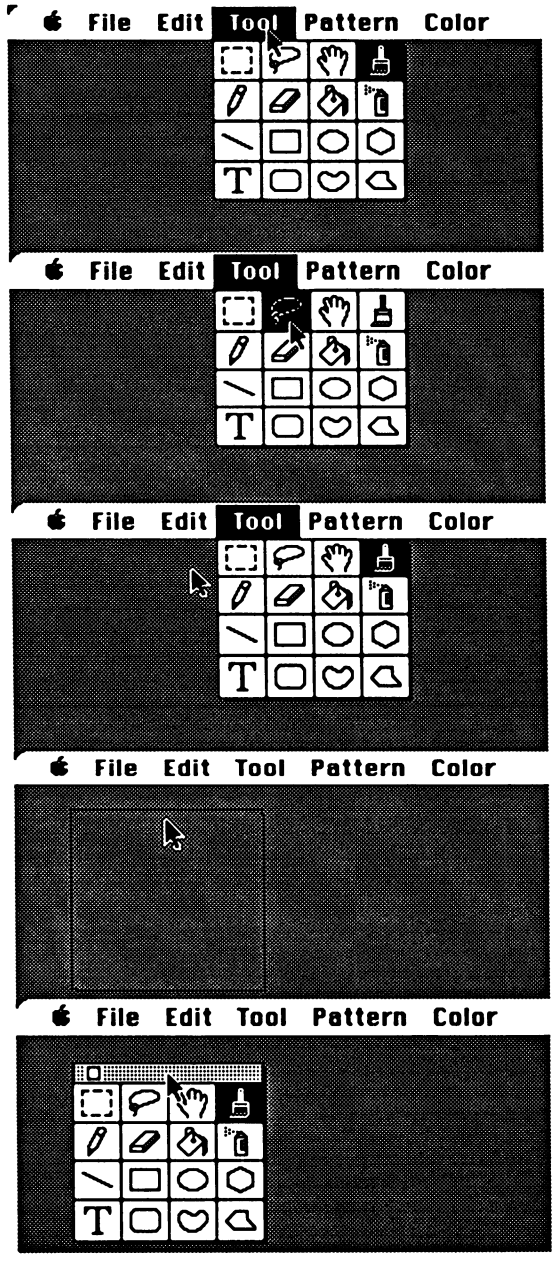


Fig. 1 Tear off palette menus that float on top!

© The Definitive MacTutor, Vol. 4

Before we get to the code though, we'd like to cover some user interface considerations, the proper use of menu and window definitions, some guidelines for using our special MDEF, and a general explanation of how the whole thing works.

User-Interface Thought Police on Patrol

We're going to pretend we're Apple Computer and tell you how tear-off menus and floating palettes *should* behave. After all, we *are* writing this with plural pronouns.

Before a menu is torn off it must act like a normal menu. The currently selected item must be highlighted when the menu is drawn. The current item must be unhighlighted and another item must be highlighted as the mouse is tracked through the menu. When the mouse button is released within the menu, the highlighted item must flash and become the currently selected item.

When the mouse is more than 15 pixels outside the menu, a gray outline of the palette must track the mouse until the button is released or the mouse moves back within the menu or menu bar. The mouse must be 'stuck' to the top center of the gray outline, just as if it were dragging a window by its title bar. If the mouse button is released outside the 15 pixel boundary, a floating palette must appear within the gray outline. See Figure 1.

If a non-application window is the front window when the menu is torn off, the top-most document and any other floating palettes must be brought in front of that non-application window. The torn-off menu must then become the top-most palette. The top-most document or other floating palettes must never be unhighlighted when a menu is torn off. See Figure 2.

Floating palettes must always remain above documents in their own layer and they must have a user-selectable order in that layer. If the only visible document is closed, all of the palettes must remain visible.

In a non-multitasking environment, whenever a non-application window is brought to the front, the top-most document and all floating palettes must be unhighlighted. See Figure 3. Under MultiFinder, whenever another application is brought to the front, the top-most document must be unhighlighted and all of the visible palettes must be hidden. When the application is brought back to the front, all hidden palettes must be made visible and the top-most document must be highlighted.

Whew! And you thought Apple was stuffy in their "Human Interface Guidelines: The Apple Desktop Interface," a recent must-read publication from Addison-Wesley. If you've got a copy handy, turn to pages 90 through 92 and see how close our definitions match.

Hmmm. We seem to be much more specific about certain details. Why? Well, after using HyperCard and the new MacPaint we began to notice some of the quirks we mentioned earlier. Some behavior just didn't seem to conform to the standard Macintosh user interface. Like not unhighlighting floating palettes when a desk accessory is brought to the front. Which then is active, the desk accessory or the application? If the windows don't overlap, the user can easily become confused.

Tear-off menus and floating palettes shouldn't break any of the other normal interface guidelines. Think of them as an extension or super-set of the regular Mac interface. Also, always remember Scott Knaster's rule, "Apple is not God." That's why we don't require the user to hold down the infamous command key to tear off a menu. That rule reeks of modality. Why make the convenience of tearing off a menu with the mouse more inconvenient by having to use the keyboard? Obviously Bill Atkinson feels the same way. And remember, HyperCard *is* system software.

Of course, if you hold down the command key, our menus will still tear off.

And don't be confused by Apple's insistence that "if a single application can have more than one menu torn off at a time, the application must determine their order of precedence." Of course, floating palettes must always be in some order. But the application should let the user determine that order by clicking the mouse in the title bars of the palettes. It's still a determined order of precedence, just not fixed.

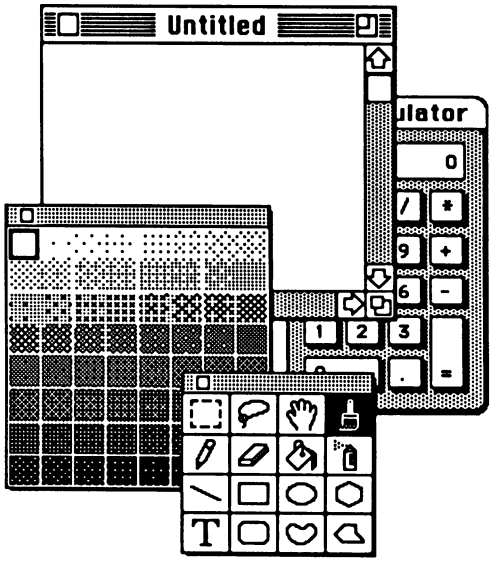


Fig. 2 Our palette floats above the document window

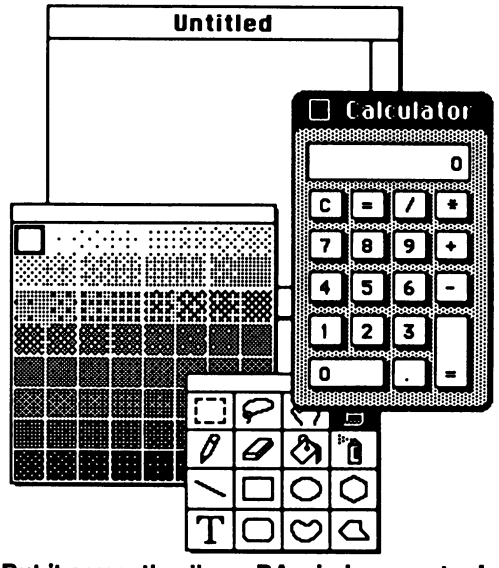


Fig. 3 But it correctly allows DA windows on top!

We did add one extension to our application's user interface and it has nothing to do with tear-off menus or floating palettes. It's something that we think the Mac has needed for quite awhile. In fact, Mike Schuster performed a similar extension with Adobe Illustrator. This is simply to allow the user to send a window to the back by holding down the option key and clicking on the window's title bar. And we extended this extension to include the palettes. A floating palette can be sent underneath all of the other visible palettes in the same manner.

Also, just like the Finder, if the option key is held down while closing one of our application windows, all open windows are closed.

Writing the Darn Thing

So now that we've decided what we want to do, how do we go about doing it?

Well, we could use Andy Hertzfield's method but it's very complex. Of course, since Andy's a genius you'd expect that. Andy decided to patch several Window Manager traps and lie to the ROM about what's actually going on in the Window Manager port. He also plays hide-and-go-seek with the Menu Manager and a lot of other system software. We decided not to get that low-level with our code since we're only planning to tear off a few of our application's menus.

First, to create a menu that will tear off, a menu definition must be written. If that torn-off menu will display a palette that looks different from a standard window, then a window definition must be written. Daryl Lovato showed us this much. But to make those palettes float above application documents, without patching any traps, substitutes for certain Window Manager routines must be implemented within the application code.

Of course, we could just rewrite the entire Window Manager as a library and link it with our code. Anyone have a *lot* of time on their hands? Just kidding, we'll get to this later anyway.

Since the first job seems to be writing menu and window definitions (MDEFs and WDEFs), we can just jump right in. Right? Well ... there's a little disagreement, and even more confusion, about how these MDEFs and WDEFs should be put together.

MDEFs and WDEFs are called by the Macintosh ROM to do things like draw a menu or highlight a window. The System file and newer versions of the ROM contain the standard definitions as separate resources. Apple recommends that whenever you have to display non-standard menus, windows, etc., you should write and compile these routines as separate resources and move them into your application.

In reality almost nobody, including Apple, does it this way.

Most folks create the definitions as a part of their normal application code and at runtime make the ROM call their code as if it existed in a separate resource. However, there is a right way and a wrong way to do this. Let us explain MDEFs first.

Using a Menu Definition

One of the fields of a MENU resource is the procID. This is a short integer containing the resource ID of the MDEF to be used by the Menu Manager to draw, highlight, etc., the menu. When the menu is inserted into the menu list, the Menu Manager replaces the procID and the following filler field, also a short integer, with a handle to the MDEF. Thus one long, the handle, replaces two shorts, the procID and filler, in the menuProc field of a MenuInfo record.

Have you noticed that the resource seems to be arranged a bit differently than the MenuInfo record? Relax, it works fine because the whole thing is really the same size.

Normally the procID contains 0 to refer to the default MDEF in the system. Programmers who create their own separate MDEF code resources place the ID of their MDEF in a MENU resource's procID. This is the Apple approved method. It works too, but it's more difficult for non-standard MDEFs to be completely separate since they usually contain information specific to the application which must be drawn or highlighted. Some method of communicating with the application must exist. That's why so many programmers leave the menu definition in their application code, so it can access their own global variables and such.

How is that done? One method is too leave your MENU resource with a 0 in its procID, insert the menu into the menu list, and then place a handle to a routine in your application code in the menuProc field that will behave like a normal MDEF. The tricky part is creating a handle to your own code. This is where some programmers try to be clever and wind up doing something very dangerous with Macintosh memory management. Here's the dangerous evil mutant method:

```
DangerousMethod() {
 MenuHandle myMenuHdl;

 /* Get a handle to the menu
 and insert it into the menu list. */
 myMenuHdl = GetMenu(MENU_ID);
 InsertMenu(myMenuHdl, 0);

 /* Point menuProc at empty block on the heap. */
 (*myMenuHdl)->menuProc = NewHandle(0);

 /* Arrggh!
 Aim the block's master pointer at your code!? */
 *(*myMenuHdl)->menuProc
      = (Ptr) MenuDefintionFunction;

 /* Force the menu size to be recalculated
 after it's inserted. */
 CalcMenuSize(myMenuHdl);

 DrawMenuBar();
}
```

Never, never, never change the contents of a master pointer this way! As soon as you do, the block to where it formerly pointed no longer exists as far as the memory manager is concerned. And worse still, the block to where it now points is inside another block. Strange and evil things can begin happening to your heap. For more information on this phenomenon, consult Tech Note #117, pages 17-19. Apple has some very strict precautions against using this technique.

So how *do* you get a handle to your code? It's so simple it's scary. Put a handle to a routine that will jump to your application

code in the menu's menuProc field. What? Another routine? Aren't we going in circles here? Nope, just create it at runtime. Let's define a new type of structure and a new constant:

```
typedef struct JumpRecord {
 short instruction;
 void (*function)();
} JumpRecord, **JumpHandle;

#define JMP_INSTRUCTION 0x4ef9
```

Figured it out? Right! All we have to do is allocate one of these records as a relocatable block on the heap, put the value of a 68000 jump instruction in the first field and the address of the menu definition in the second field. Voila! When the ROM looks at this it will see:

```
JMP MenuDefinitionFunction
```

Sneaky? Yeah, but it's one safe and sane method that's also perfectly legal. Here's how you set it up:

```
SafeAndSaneMethod() {
 MenuHandle myMenuHdl;
 JumpHandle menuDefJump;

 /* Get a handle to the menu
 and insert it into the menu list. */
 myMenuHdl = GetMenu(MENU_ID);
 InsertMenu(myMenuHdl, 0);

 /* Allocate a JumpRecord on the heap. */
 menuDefJump
      = (JumpHandle)NewHandle(sizeof(JumpRecord));

 /* Init JumpRecord to point to your code. */
 (*menuDefJump)->instruction = JMP_INSTRUCTION;
 (*menuDefJump)->function = MenuDefinitionFunction;

 /* Point menuProc at JumpRecord on the heap. */
 (*myMenuHdl)->menuProc = (Handle) menuDefJump;

 /* Force the menu size to be recalculated
 after it's inserted. */
 CalcMenuSize(myMenuHdl);

 DrawMenuBar();
}
```

Now some might argue that this forces the application to become a compiler. In a sense that's true, but the same argument could be made for any variable initialization done by an application. Hey, it works. And it works well.

There's one important thing you need to remember when you're going to have the ROM call your application code. Never let your code move! If you give the ROM a pointer, it better point to the real thing. This means that the menu definition better reside in a code segment that's always available and *always* locked. Place it in your main segment and you shouldn't have to worry.

There's a variation on the safe and sane method that's definitely worth mentioning. You don't have to create the JumpRecord at runtime. Instead, you can create an MDEF resource that's the same size as the record, have the ROM read it in, and initialize it the same way. And it even saves a few lines of code. Other than that, there's no functional difference at all. Here's an example:

```
OtherMethod() {
 JumpHandle menuDefJump;
 MenuHandle myMenuHdl;

 /* Get a handle to the MDEF. */
 menuDefJump
      = (JumpHandle)GetResource('MDEF', MDEF_ID);

 /* If the first word of the MDEF resource
 contains 0x4ef9 then there's no need to
 initialize the jump instruction here. */

 /* Initialize the MDEF to point to your code. */
 (*menuDefJump)->function = MenuDefinitionFunction;

 /* Get a handle to the menu
 and insert it into the menu list. */
 myMenuHdl = GetMenu(MENU_ID);
 InsertMenu(myMenuHdl, 0);

 /* There's no need to force the menu size
 to be recalculated. */

 DrawMenuBar();
}
```

By the way, if you place a different number in the procID of a MENU resource, a corresponding MDEF must be available at runtime or the Menu Manager is particularly unforgiving; i.e., a system error will occur.

Using a Window Definition

WDEFs usually don't need to communicate with an application because they just draw the window frame. The contents of a window are (and should) always be handled by the application itself in its update function. Of course that doesn't mean you can't include a window definition in your application code. It's just not as necessary.

If you're going to write a separate WDEF, like we did, you still have one thing you must do: pass the correct procID to NewWindow(). "Inside Macintosh Volume I," page 298, explains this whole process in detail. Since the standard WDEF's resource ID is 0, most programmers don't know they're calculating the procID like this when they create a window:

```
procID = (16 * resource ID) + variation code
```

Obviously (16 * 0) + noGrowDocProc would still equal noGrowDocProc, so most of us never notice this. Here's an example in context:

```
UseRealWDEF() {
 WindowPtr myWindow;

 myWindow = NewWindow(p, bounds, title, visible,
      (16 * WDEF_ID) + noGrowDocProc, behind,
      goAwayFlag, refCon);
}
```

However, if your window definition is going to reside in your application code, you'll have to do things a bit differently. If you use the six byte jump table WDEF in a resource (like we

did with the MDEF), you're going to have to initialize it *first* before you pass its ID to NewWindow(). Why? Because the WDEF code is executed when the window is created. If you don't have the address to the function stored in the jump table, the ROM will jump to the flaming pits of Hell. Not a nice thing to happen. So here's how it can be done correctly:

```
UseJumpTableWDEF() {
 JumpHandle windowDefJump;
 WindowPtr myWindow;

 /* Get a handle to the WDEF. */
 windowDefJump
      = (JumpHandle)GetResource('WDEF', WDEF_ID);

 /* If the first word of the WDEF resource
 contains 0x4ef9 then there's no need to
 initialize the jump instruction here. */

 /* Initialize the WDEF to point to your code. */
 (*windowDefJump)->function
      = WindowDefinitionFunction;

 /* Now it's safe to create the window. */
 myWindow = NewWindow(p, bounds, title, visible,
      (16 * WDEF_ID) + noGrowDocProc, behind,
      goAwayFlag, refCon);
}
```

If you'd rather create the jump table at runtime (like we also did with the menu), there is one limitation: your window definition won't be able to respond to a wNew message (See "Inside Macintosh Volume I," pages 298-302). This is because there isn't any real resource to pass to NewWindow() in the procID, and a wNew message is executed only when the window is created. A real chicken-before-the-egg problem. But, most of the time this is no big deal, so here's how it's done:

```
UseOnlyAJumpTable() {
 WindowPtr myWindow;
 JumpHandle windowDefJump;

 /* Create window, but don't make it visible. */
 myWindow = NewWindow(p, bounds, title, FALSE,
      noGrowDocProc, behind, goAwayFlag, refCon);

 /* Allocate a JumpRecord on the heap. */
 windowDefJump
      = (JumpHandle)NewHandle(sizeof(JumpRecord));

 /* Init JumpRecord to point to your code. */
 (*windowDefJump)->instruction = JMP_INSTRUCTION;
 (*windowDefJump)->function = WindowDefintionFunction;

 /* Point windowDefProc at JumpRecord on heap. */
 ((WindowPeek) myWindow)->windowDefProc
      = (Handle) windowDefJump;

 /* Now you can make the window visible. */
 ShowWindow(myWindow);
}
```

Whatever you do, don't try the evil mutant method again! Here's a *real* dangerous way to get your code executed:

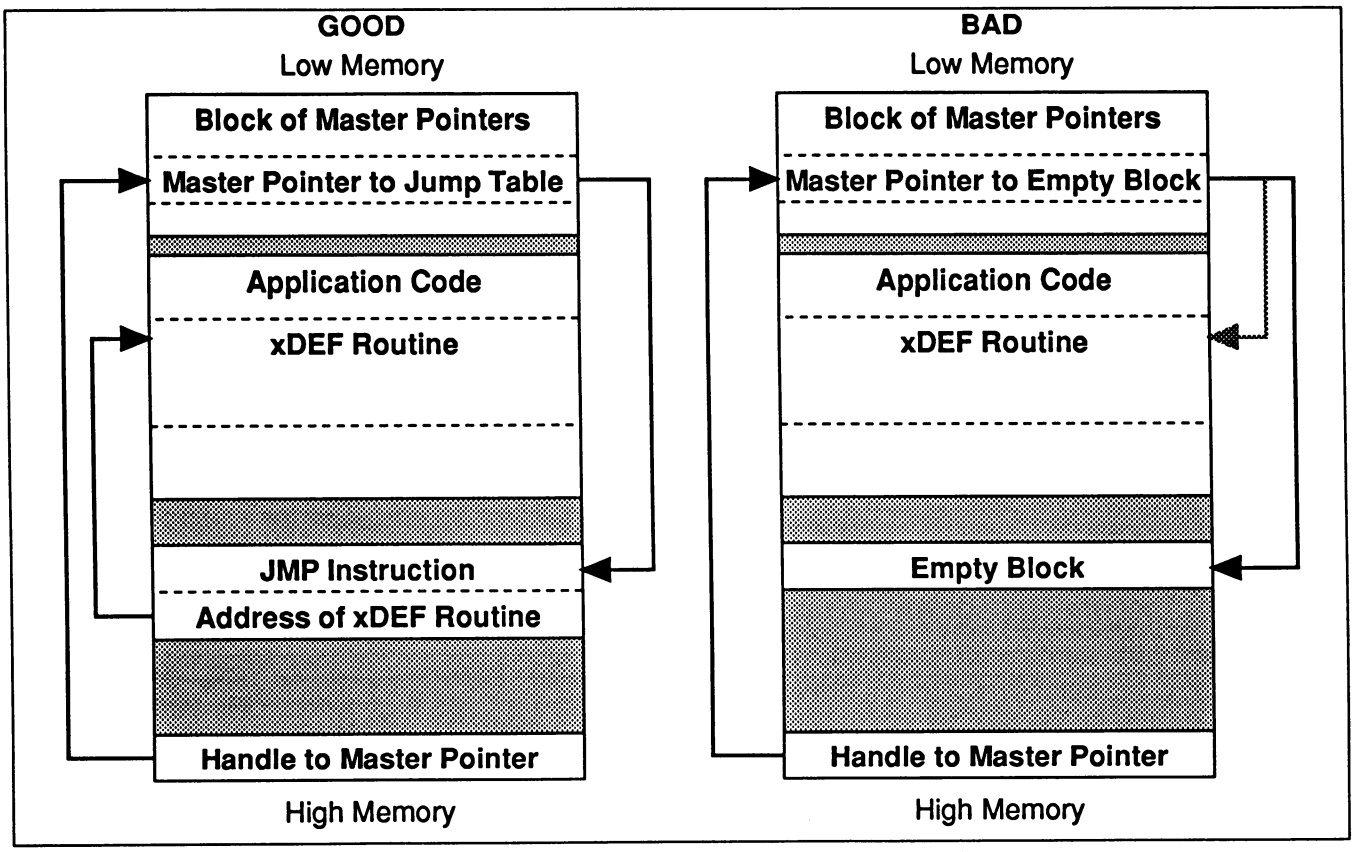


Fig. 4 The right and wrong of definition routines

© The Definitive MacTutor, Vol. 4

```
DangerousMethodForWindows() {
 WindowPtr myWindow;

 /* Create window, but don't make it visible. */
 myWindow = NewWindow(p, bounds, title, FALSE,
      noGrowDocProc, behind, goAwayFlag, refCon);

 /* Point windowDefProc at empty block on heap. */
 ((WindowPeek) myWindow)->windowDefProc
      = NewHandle(0);

 /* Arrggh!
 Aim the block's master pointer at your code!? */
 *((WindowPeek) myWindow)->windowDefProc
      = (Ptr) WindowDefintionFunction;

 /* Make the window visible. */
 ShowWindow(myWindow);
}
```

Never use master pointers like this! Enough said. Figure 4 diagrams the right and wrong way to point to your code.

Our Solution

So, there are many methods to connect an application to a definition function. Certain methods to avoid entirely. But which is best? There is no best way; it depends entirely upon what you're trying to do. Still, Apple suggests applications that ship should contain these definitions as separate resources. The key word there is ship.

Also remember that Apple doesn't always follow their own advice. And it's advice, not the law. The menu and window definitions used in HyperCard and the new MacPaint (which was developed at Apple before Claris even existed) are not contained in separate MDEF and WDEF resources. However, Bill Atkinson and other folks in Cupertino know better than to use the dangerous methods described above.

The reality of it is that during development it's much easier to keep the definition in your application code, especially if you use LightspeedC. Moving a separately compiled project in and out of another resource file is a real pain. If you write the definitions correctly, it's not that much trouble to move them into a separate code resource at the end of your development cycle, providing you're not doing the last compilation the day before you shrink wrap your product.

When we told a few folks in Apple Tech Support that we were going to write this article, they encouraged us to code our example with a separate MDEF and WDEF. Everyone could use them then. Not only would our source code be handy, but folks could just use our xDEFs like libraries. Universal code. What a great idea!

That was easy with the WDEF, but how *do* you share global data with a separate MDEF? Well, we had to get just a little bit sneaky ...

Communicating with TearOffMDEF

So you've probably skipped ahead and looked at the code by now. Confused? Serves you right for skipping ahead. It's really simple. All we do is provide the basic tear-off mechanism and the logic to highlight and unhighlight menu items. You write all the hard stuff in your application.

What? Isn't the MDEF separate? Sure, but it's not clairvoyant. Somebody has to draw the menu. TearOffMDEF passes application specific requests made by the ROM back to the application. The application must first create the palette window to be torn off, provide three functions for the MDEF to use, and create and initialize a structure that will be used to contain global information.

Our TearOffMDEF communicates with an application via a TearOffMenuGlobals structure in a TOMG resource. What's a TOMG resource? Well, it's a 28-byte block with nothing but zeroes in it. But we'll fill it up quickly. At runtime, TearOffMDEF uses the menuID of the MenuInfo record passed to it by the Macintosh ROM to find a TOMG resource of the same ID.

Okay, let's make sure we're clear on this. A MENU resource has an ID. One of its fields also contains an ID. They're usually the same, too. Obviously this sub-ID can be found with a handle to the menuInfo record, because the menuInfo is basically a duplicate of the original MENU resource. TearOffMDEF uses this sub-ID to find a TOMG resource. Why? This way the application and the MDEF can share global information without ever referencing a compiled offset of address register 5. They don't have to share any of the same source code files. They just need to define the structures the same way.

Yes, we thought it was pretty clever.

Here's the way the global structure should be declared by the application:

```
typedef struct TearOffMenuGlobals {
 void (*drawMenuProc)();
 short (*findItemProc)();
 void (*hiliteItemProc)();
 SysEnvRec *environment;
 WindowPtr paletteWindow;
 Point position;
 short currentItem;
 Boolean itemHilited;
} TearOffMenuGlobals, *TearOffMGlobalsPtr, **TearOffMGlobalsHdl;
```

There must be one of these structures for each of the menus that will be torn off. The TearOffMenuGlobals structure contains eight elements:

drawMenuProc: A function pointer to a Pascal-type procedure which draws the contents of the menu in global coordinates within the WMgrPort.

findItemProc: A function pointer to a Pascal-type procedure which returns the number of the menu item where the mouse is currently located.

hiliteItemProc: A function pointer to a Pascal-type procedure which highlights or unhighlights a given menu item.

environment: A pointer to a SysEnvRec used to test if the 64K ROMs are present.

paletteWindow: A WindowPtr used to calculate menuWidth, menuHeight, and the structure boundary from the window's portRect.

currentItem: A short int containing the number of the currently highlighted menu item.

position: A Point passed to the application from TearOffMDEF indicating the top, left point, in global coordinates, of a torn-off menu.

itemHilited: A Boolean used internally by TearOffMDEF.

All of these elements, except position and itemHilited, must be initialized by the application before a tear-off menu is inserted into the menu list.

Remember, the application is completely responsible for the appearance of the menu. A set of the three following procedures must be declared in the application for each tear-off menu:

```
pascal void DrawMenuProc(destRect)
Rect *destRect; /* The menu rectangle in the current
                grafPort where the application should
                draw the menu. */

pascal short FindItemProc(mousePt)
Point mousePt;  /* The point in which the mouse is
                currently located, relative to the
                top, left of the menu rectangle. */

                /* This routine must return the
                number of the menu item where the
                mouse is currently located. */

pascal void HiliteItemProc(destRect, item, hilite)
Rect *destRect;  /* The menu rectangle in the current
                grafPort. */
short item;     /* The number of the item to be
                highlighted or unhighlighted. */
Boolean hilite; /* A flag that determines the state
                of the highlighting. True means
                highlight the item, false means
                unhighlight. */
```

Inside TearOffMDEF

If you want to take a closer look at our TearOffMDEF code, pay particular attention to a few points.

We must refer to QuickDraw's globals in a peculiar fashion. Why? Since we're not an application, we can't call InitGraf() to initialize any compiler offsets to address register 5. We have to set up a pointer to these globals via the low-memory location CurrentA5 and refer to them by explicit dereference of that pointer. A bit inconvenient, but it means that when you want to draw in white it doesn't come out looking like chicken scratches.

When we track the mouse to correctly highlight items, we make sure the ROM will correctly flash the item when the mouse is released. Getting the menu to flash is more than just making the algorithm in your code function in a certain way. Specifically, you must insert enough fake items in the MENU resource so that the Menu Manager thinks the item you chose exists when it calls CountMItems(), otherwise it won't flash it.

Furthermore, our code expects the Menu Manager to tell us the mouse is in position (0, 0) when it's trying to flash the item. An undocumented feature, true, but it hasn't changed yet. This doesn't happen if the menu is called via PopUpMenuSelect() (and no, we don't know why), but since we don't think pop-up menus should be torn off, we don't even worry about it.

And how does the menu tear off? If the Menu Manager calls TearOffMDEF and the mouse is outside the menu rectangle by more than 15 pixels, our DragMenu() routine is called. Simple. DragMenu() follows the mouse around (just like DragGrayRgn() in the ROM) with a gray outline of the menu. Notice that the outline doesn't flash like HyperCard or the new MacPaint. DragMenu() retains control until the mouse button is released or the mouse returns to the menu rectangle or menu bar. If the mouse returns the code returns. If the mouse button is released outside the menu rectangle, we set the item selected to -1 and place the top, left corner of where the palette should be drawn in the position field of the appropriate TearOffMenuGlobals structure.

Using TearOffMDEF in Your Application

If you intend to use TearOffMDEF, you're going to have to move the MDEF code resource file into your application. If you're using LightspeedC, just paste it into the common resource file with ResEdit. If you're translating this to MPW C, then you can use Rez to put it all together at once.

By the way, there is one very important thing to remember when you create an MDEF (or another kind of xDEF), always make sure it's resource flags are set to non-purgeable. Why? The ROM doesn't like it when a code resource gets purged at runtime. Any heap compaction can force this to happen.

Also, there's another REAL thrill you need to work around if you compile this MDEF. It's a conflict with LightspeedC and our favorite ROM. As we all know, the Menu Manager passes the menu item to be manipulated to an MDEF by address. Usually the item is located on the stack. But when the Menu Manager repeatedly calls your code to cause a menu item to flash, it passes ToolScratch as the address of the item. Why? Who knows. The conflict occurs because LightspeedC stores the address of the separate code resource (in this case, the MDEF) in ToolScratch as soon as that code is called by the ROM. Why? This allows a programmer (who doesn't want to use in-line assembly) to find out exactly where the code exists in memory by simply looking in ToolScratch. A perfectly legal and very clever use of that low-memory global.

Believe it or not, the ROM's use of ToolScratch is definitely Apple's problem. When we told Paul Mercer in Apple Tech Support about this, he even filed an official bug report. Is that service or what?

To work around the problem, Michael Kahl, the author of LightspeedC, says he may change his glue for code resources in the soon-to-be-released version 3.0 of his compiler. Until then, both Apple and Mike suggest patching the MDEF after it's compiled. A real pain we know, but use ResEdit or FEdit to change hex string '21c809ce' at hex offset 14 in the MDEF resource to '4e714e71.' This substitutes two 'nop' instructions for a 'move.l a0,ToolScratch.'

Mike Kahl says you can also modify your copy of the compiler so it never generates these instructions when the code resource is compiled. The glue in versions 2.13 and 2.15 is located in code resource number 11, also at hex offset 14. The patch is identical. Remember, do it to a *copy*.

Let us repeat that the segment in which your application's procedures are declared *must* remain locked at runtime, the ROM will not chase your procedures around. Place it in your main segment and you shouldn't have to worry.

Remember the application must move the palette window and make it visible when MenuSelect() returns -1 from a tear-off

menu. We'll explain how we keep the little beggars floating in just a bit.

Also, when the user moves the palette or clicks the mouse inside it, some response is necessary. If you want to emulate the Menu Manager and its drawing, finding, and highlighting of menus, your application must do this itself for floating palettes. We went the easy route and just used the same three routines again that are called by the MDEF. They're already in the application code, right? We're lazy and we highly recommend this lack of effort.

Inside PaletteWDEF

Our PaletteWDEF is pretty straightforward. It creates windows almost identical to those in HyperCard or the new MacPaint. The main difference being PaletteWDEF will respond to a Window Manager request to unhighlight the title bar. Also, the drop shadow created by PaletteWDEF more closely matches a menu's drop shadow than an ordinary window.

The only real tricky thing in the code is where we create the pattern to fill the title bar. Why? Windows are drawn into the Window Manager port. The origin is always in the top, left corner of the screen. Patterns are always aligned to the origin. Depending on where the top, left corner of the window will be, the pattern may not align when the window is drawn.

Apple's standard WDEF, originally written by Andy Hertzfield, uses a rather undocumented QuickDraw global called patAlign to make sure everything looks correct. If you look at the interface files to most high-level language compilers available for the Macintosh, you won't even find this field defined. It *is* in the MPW assembler equates though.

Even though we defined our own QuickDraw structure, we decided not to use a global not even discussed in any Apple documentation that we could find. Instead, we create the pattern on the fly. It's a very regular pattern, each horizontal line contains either 55, 55, 55 etc. or AA, AA, AA etc. After we wrote this 'hack' we decided to disassemble HyperCard and see how Bill Atkinson did it. You guessed it, the same way.

Keeping Palettes Above Documents

The first thing you need to know about palettes to keep them floating is where they are and where the top-most document is. Specifically, where in the window list. Window list? Yes, the Window Manager keeps a linked list of all windows beginning in a low-memory global called WindowList. It points to the first window and it's very handy. In fact FrontWindow(), in the Mac ROM, steps through this list to find the top-most visible window and then returns a pointer to it.

We wrote our own version of FrontWindow() called LocateWindows(). It doesn't return a pointer — instead it initializes four global variables called TopWindow, TopPalette, BottomPalette and TopDocument. It's called once during the beginning of our event loop. But why do we need to know all this?

If your application is only going to use one document window, then perhaps you can keep a pointer to it in a global variable and test to see if it's visible or not. And if you only have one palette, or a fixed number of palettes, you can maintain a series of pointers to them. You can even create all of these windows at once so you know what order they're in. Right?

Well, what if somebody uses QuicKeys or the public domain Windows DA to change the order of the windows? And if you *want* your palettes to have a dynamic order or you need more than one document then you're going to have to keep track of a few things. These four globals are the minimum you need to know about the window list.

When a document window is created it can appear behind the bottom palette only if you know which window *is* the bottom palette. Look at our example code carefully and notice the care and feeding of these globals. Every operation to keep palettes floating above documents depends on LocateWindows().

And initializing these four globals isn't the only thing LocateWindows() does. If it finds a non-application window between the palettes and top-most document, it moves that window. If it finds a document above one of the palettes, it moves it below the bottom palette.

Usually FrontWindow() is also called by an application to tell if that application is active. We wrote a small function called ActiveWindow(), to which we pass a window pointer and test it to make sure it is either the TopDocument or one of the palettes.

Now the hard part.

Normally if you click in a window, it's brought in front of all other windows. SelectWindow() is always used to do this. But you can't call SelectWindow() for a document if a palette is visible. How do you keep it from being brought to the front? Use SendBehind() instead. Right. Just send the document behind the BottomPalette.

Look carefully at our code to see how we did this in a function called DoSelectWindow(). You'll also notice that we use two low-level Window Manager routines called CalcVisBehind() and PaintOne() after we move the document. Why? If a window is sent behind a window that is above it, the Window Manager must redraw parts of the affected windows. We speed up this process by calculating the minimum area to be redrawn. See "Inside Macintosh Volume I," page 286, for details on this process.

Another ROM call to avoid is DragWindow(). Annoyingly, DragWindow() will always bring a window to the front if the window pointer passed to it is not already in front. You can hold the command key down while you drag a window to keep this from happening, but there's no clean way to 'depress' this key from within your code. Instead, look at DoDragWindow() and Drag() in our code. We had to completely rewrite this ROM routine. DoDragWindow() handles all of the logic of when and where to drag, and Drag() actually moves an outline of the window around the screen.

Are you seeing a pattern yet? Right. To suspend palettes above documents, never call a function which will directly move a document to the front if a palette is visible.

Still, problems with window highlighting, the appearance of controls, and the grow icon can occur while doing our juggling act with the Window Manager. Basically, you can't depend on the ROM to always activate or deactivate a document — it's not *really* the front window. "Inside Macintosh Volume I," page 280

contains a discussion of two low-memory variables called CurActivate and CurDeactive. Clever manipulation of these globals can direct the Event and Window Managers to activate or deactivate something other than the front window. It's all legal and it works just fine.

But Apple may change these low-memory globals one of these days. With the advent of MultiFinder and the Layer Manager, the Window Manager has been changing. Just to be safe, we've developed a different technique to do this that doesn't write to these locations. It's a bit of a pain, but it works. Essentially, we must explicitly highlight windows, activate controls, and redraw grow icons when documents are moved within the window list. Take a look at two of our routines called ActivateDocument() and HiliteUserWindows() — they're pretty straightforward.

Remember, the whole process of floating a palette is a charade. First you fool the ROM, and then you can fool the user.

How to Read the Code

First we'd like to acknowledged our debt to the folks in Apple Tech Support for providing us with some helpful clues. Also, many thanks to Mike Kahl and the rest of the gang on Compuserve's LVTForum, home to many LightspeedC and Pascal nerds (Are you a THINKer? Just type GO THINK when you're on CIS). And finally, thanks to David Smith for believing we could pull it off.

We've tried to give as complete an example as possible. Although our code can be used almost like libraries, the whole idea is to share needed information with the rest of the Mac community. And to show off, of course. If you read this and come up with some better solutions or (horror!) bugs, please let us know.

You might have already noticed that we wrote all of our code in C. Why? Sometimes we *want* to send the wrong type somewhere. Of course, we don't recommend this, we just do it. At least we didn't use any in-line assembly.

Actually, we designed the code to be easily translated into Pascal. Too bad if you're using COBOL though. We didn't use too many special C 'tricks' that aren't obvious anyway. If you look at the application code, you may notice that it's rather repetitive in sections that deal with updating or drawing into palette windows. In our own project we're developing we use arrays of function pointers indexed with a palette's ID number (stored in the window refCon field) to make everything much shorter and more elegant. We thought this would probably confuse Pascal programmers though.

Although we wrote this in LightspeedC, the resources are listed in MPW Rez format. Sorry, but we never use RMaker. It's just not powerful enough.

This whole thing is a monster. Our advice is buy the source code disk from David. It's actually contained in three LightspeedC project files. Here's a breakdown:

TearOffMDEF.project

TearOffMDEF.c The menu definition.

PaletteWDEF.project

(No need to link with MacTraps)

PaletteWDEF.c The window definition.

TearOffPalette.project

(All of these go in the main segment)

TearOffPalette.c The main program.

Interface.c Adjusts the use of color at runtime.

Error.c Error and grow zone function.

Window.c Creation and updating of windows.

Dialog.c About dialog and utilities.

Menu.c The routines called by the MDEF.

Palette.c The code that floats the palettes.

(Place this in a separate segment)

Initialize.c Initialization and creation of palettes.

(The include files are used by all of the sources)

Constants.h include files and our definitions.

Variables.h Global variables.

(MPW Rez format resource file)

TearOffPalette.project.r

(Move the MDEF and WDEF into this file with ResEdit)

TearOffPalette.project.rsrc

Finally, if you program in C, you had better try to maintain a style of some kind. If you don't, nobody will ever be able to make heads or tails of your code, including you. We've tried to use classic indentation, but we'd like to clarify our nomenclature.

FunctionName()	Always an initial capital.
GlobalVariable	Just like the functions.
localVariable	The first letter is always lower case.
A_CONSTANT	All capitals, underscores dividing words.

Of course many of Apple's globals and constants don't follow our rules. Oh well. It might not make sense to you, but at least we're consistent. On to the code!

```
/*——TEAROFF  PALETTE

  version 1.0
  by Don Melton and Mike Ritter

  Copyright © 1987, 1988 by Impulse Technologies,
Inc., all rights reserved.

  Filename:          Constants.h
  Font:              Courier, 9 point
  Tab setting:       2
  Compiler:          LightspeedC 2.15
  Project type:    APPL
  Creator:           TOPD */

/*————————————INCLUDE DEFINITIONS */

#include "Color.h"
#include "ColorToolbox.h"
```

```
#include "ControlMgr.h"
#include "DeskMgr.h"
#include "DialogMgr.h"
#include "EventMgr.h"
#include "FileMgr.h"
#include "FontMgr.h"
#include "MacTypes.h"
#include "MemoryMgr.h"
#include "MenuMgr.h"
#include "OSUtil.h"
#include "PackageMgr.h"
#include "Quickdraw.h"
#include "ResourceMgr.h"
#include "SegmentLdr.h"
#include "StdFilePkg.h"
#include "TextEdit.h"
#include "ToolboxUtil.h"
#include "WindowMgr.h"

/*------------------------------SYSTEM CONSTANTS */

#define nil 0
#define NULL 0
#define zoomDocProc 8
#define zoomNoGrow 12

/*------------------------------ENVIRONMENT */

#define VERSION_REQUESTED 1

#define WAIT_NEXT_EVENT_TRAP_NUMBER 0x60
#define UNIMPLEMENTED_TRAP_NUMBER 0x9f

/*------------------------------MEMORY */

#define MEMORY_BUFFER_SIZE 0x8000 /* 32K. */

/*------------------------------EVENTS */
#define SLEEP_DURATION 50

/*------------------------------WINDOWS */

#define VISIBLE true
#define NOT_VISIBLE false
#define BRING_TO_FRONT (WindowPtr) -1
#define SEND_BEHIND nil
#define GO_AWAY_BOX true
#define NO_GO_AWAY_BOX false
#define NO_REFCON nil

/*------------------------------PALETTES */

enum {
  TOOL_PALETTE,
  PATTERN_PALETTE,
  COLOR_PALETTE,
  DOCUMENT_WINDOW
};

#define PALETTE_COUNT 3

#define PALETTE_WDEF_ID 128
#define TEAROFF_MDEF_ID 128

/*------------------------------TEAROFF MENU GLOBALS */

typedef struct TearOffMenuGlobals {
  void (*drawMenuProc)();
  short (*findItemProc)();
  void (*hiliteItemProc)();
  SysEnvRec *environment;
  WindowPtr paletteWindow;
  Point position;
  short currentItem;
  Boolean itemHilited;
} TearOffMenuGlobals, *TearOffMGlobalsPtr, **TearOffMGlobal-
sHdl;

#define TEAR_OFF_MENU_GLOBALS_TYPE 'TOMG'

/*------------------------------TOOL PALETTE */

#define TOOLS_ACROSS 4
#define TOOLS_DOWN 4
#define TOOL_COUNT ((TOOLS_ACROSS * TOOLS_DOWN) + 1)
#define DEFAULT_TOOL 4
#define TOOL_PICT_ID 128

/*------------------------------PATTERN PALETTE */

#define PATTERNS_ACROSS 8
#define PATTERNS_DOWN 8

#define PATTERN_COUNT ((PATTERNS_ACROSS
      * PATTERNS_DOWN) + 1)

#define PATTERN_ITEM_WIDTH 16
#define PATTERN_ITEM_HEIGHT 16

#define DEFAULT_PATTERN 1
#define PATTERNS_ID 128

/*------------------------------COLOR PALETTE */

#define COLORS_ACROSS 16
#define COLORS_DOWN 16
#define COLOR_COUNT ((COLORS_ACROSS * COLORS_DOWN) + 1)
#define COLOR_ITEM_WIDTH 8
#define COLOR_ITEM_HEIGHT 8
#define DEFAULT_COLOR 1

/*------------------------------DOCUMENTS */

#define TITLE_BAR_HEIGHT 18
#define SCROLL_BAR_SIZE 16
#define SCREEN_MARGIN 4
#define MIN_WINDOW_WIDTH 80
#define MIN_WINDOW_HEIGHT 80
#define HORIZONTAL_WINDOW_OFFSET 8
#define VERTICAL_WINDOW_OFFSET 8
#define DOCUMENT_COUNT 8
#define DOCUMENT_TITLE_STRING_ID 128

/*------------------------------MENUS */

enum {
  APPLE_MENU_INDEX,
  FILE_MENU_INDEX,
  EDIT_MENU_INDEX,
  TOOL_MENU_INDEX,
  PATTERN_MENU_INDEX,
  COLOR_MENU_INDEX
};

enum {
  APPLE_MENU_ID = 128,
  FILE_MENU_ID,
  EDIT_MENU_ID,
  TOOL_MENU_ID,
  PATTERN_MENU_ID,
  COLOR_MENU_ID
};

#define MENU_COUNT 6
#define MENU_ID_OFFSET APPLE_MENU_ID

#define TEAR_OFF_MARGIN 15
#define MOVE_PALETTE_ITEM (-1)
```

```
/*-------------------------APPLE MENU ITEMS */

#define ABOUT_ITEM 1

/*-------------------------EDIT MENU ITEMS */

enum {
  UNDO_ITEM = 1,
  CUT_ITEM = 3,
  COPY_ITEM,
  PASTE_ITEM,
  CLEAR_ITEM
};

/*-------------------------FILE MENU ITEMS */

enum {
  NEW_ITEM = 1,
  CLOSE_ITEM = 3,
  QUIT_ITEM = 5
};

/*-------------------------ABOUT DIALOG */

#define ABOUT_DIALOG_ID 128

  /*-------------------------T E A R O F F  P A L E T T E

     version 1.0
     by Don Melton and Mike Ritter

     Copyright (C)1987, 1988 by Impulse Technologies,
     Inc., all rights reserved.

     Filename:       Variables.h
     Font:           Courier, 9 point
     Tab setting:    2
     Compiler:       LightspeedC 2.15
     Project type: APPL
     Creator:        TOPD */

  /*-------------------------GLOBAL VARIABLE DECLARATIONS */

  #ifdef GLOBAL_VARIABLES

  SysEnvRec Environment;
  EventRecord Event;

  GDHandle MaxDevice;

  MenuHandle Menus[MENU_COUNT];

  TearOffMGlobalsHdl TearOffs[PALETTE_COUNT];
  WindowPtr Palettes[PALETTE_COUNT];

  WindowPtr TopWindow;
  WindowPtr TopPalette;
  WindowPtr BottomPalette;
  WindowPtr TopDocument;

  Rect ToolRects[TOOL_COUNT];
  Rect PatternRects[PATTERN_COUNT];
  Rect ColorRects[COLOR_COUNT];

  short PositionCounter;

  long Sleep;

  Boolean ClosingAll;
  Boolean Quitting;
  Boolean Finished;

  Boolean OutOfMemory;
  Boolean WNEIsImplemented;
  Boolean MouseInMenu;

  Boolean ColorMenuVisible;
  Boolean PalettesVisible[PALETTE_COUNT];

  Handle MemoryBuffer;

  #else

  extern SysEnvRec Environment;
  extern EventRecord Event;

  extern GDHandle MaxDevice;

  extern MenuHandle Menus[MENU_COUNT];

  extern TearOffMGlobalsHdl TearOffs[PALETTE_COUNT];
  extern WindowPtr Palettes[PALETTE_COUNT];

  extern WindowPtr TopWindow;
  extern WindowPtr TopPalette;
  extern WindowPtr BottomPalette;
  extern WindowPtr TopDocument;

  extern Rect ToolRects[TOOL_COUNT];
  extern Rect PatternRects[PATTERN_COUNT];
  extern Rect ColorRects[COLOR_COUNT];

  extern short PositionCounter;

  extern long Sleep;

  extern Boolean ClosingAll;
  extern Boolean Quitting;
  extern Boolean Finished;

  extern Boolean OutOfMemory;

  extern Boolean OutOfMemory;
  extern Boolean WNEIsImplemented;
  extern Boolean MouseInMenu;

  extern Boolean ColorMenuVisible;
  extern Boolean PalettesVisible[PALETTE_COUNT];

  extern Handle MemoryBuffer;

  #endif

    /*-------------------------T E A R O F F  P A L E T T E

       version 1.0
       by Don Melton and Mike Ritter

       Copyright (C)1987, 1988 by Impulse Technologies,
           Inc., all rights reserved.

       Filename:       TearOffPalette.c
       Font:           Courier, 9 point
       Tab setting:    2
       Compiler:       LightspeedC 2.15
       Project type:   APPL
       Creator:        TOPD
       Segment:        Main

    -------------------------DESCRIPTION

       TearOffPalette is a demonstration of a very portable
    method of implementing tear-off menus and floating
    palettes. This technique uses external MDEF and WDEF code
    resources. The MDEF communicates with the application via
    a TearOffMenuGlobals structure in a TOMG resource.

       See TearOffMDEF.c for an explanation of the TearOff-
```

```
MenuGlobals structure and the routines an application must
contain to use TearOffMDEF.

   See PaletteWDEF.c for an explanation of the behavior and
use of PaletteWDEF.

   TearOffPalette is designed to keep palettes floating
above documents without directly writing to low-memory
Window Manager globals.

   A Color menu and palette will be displayed when Color
QuickDraw is present and the depth of the screen allows
256 colors.

   Under MultiFinder and at any Suspend or Resume event,
TearOffPalette will hide or show all visible palettes.

   IMPORTANT! TearOffPalette.project.rsrc must contain the
TearOffMDEF and PaletteWDEF resources created by compiling
their respective source files. */

/*GLOBAL CONSTANT DEFINITIONS AND VARIABLE DECLARATIONS */

#define GLOBAL_VARIABLES

#include "Constants.h"
#include "Variables.h"

/*———————————EXTERNAL FUNCTION DECLARATIONS */

extern void Initialize();              /* Initialize.c */

extern void DoAbout();                  /* Dialog.c */

extern void AdjustInterface();         /* Interface.c */

extern void LocateWindows();            /* Palette.c */
extern short ActiveWindow();            /* Palette.c */
extern void DoSelectWindow();           /* Palette.c */
extern void DoDragWindow();           /* Palette.c */
extern void HiliteUserWindows();        /* Palette.c */

extern void ActivateDocument();         /* Window.c */
extern void UpdateDocument();           /* Window.c */
extern void SizeDocument();           /* Window.c */
extern void UpdatePalette();            /* Window.c */
extern void NewDocument();              /* Window.c */
extern void CloseDocument();            /* Window.c */

extern pascal short FindToolItem();     /* Menu.c */
extern pascal void HiliteToolItem();    /* Menu.c */
extern pascal short FindPatternItem();  /* Menu.c */
extern pascal void HilitePatternItem(); /* Menu.c */
extern pascal short FindColorItem();    /* Menu.c */
extern pascal void HiliteColorItem();   /* Menu.c */
extern void MovePalette();              /* Menu.c */

/*———————————FORWARD FUNCTION DECLARATIONS */

void DoEvent();
void MouseDownEvent();
void KeyEvent();
void ActivateEvent();
void UpdateEvent();
void SuspendResumeEvent();
void MenuEvent();

void AppleMenu();
void FileMenu();
void EditMenu();
void ToolMenu();
void PatternMenu();
void ColorMenu();
void ContentEvent();
void GrowEvent();
void InvalGrow();
void CloseEvent();

/*———————————————————————TEAROFF PALETTE */

main() {

   Initialize();
   UnloadSeg(Initialize);

   DoAbout();

   while((!Finished) && (!OutOfMemory)) {
       AdjustInterface();
       DoEvent();
   }
}

/*———————————————————————DO EVENT */

void DoEvent() {
Boolean result;

   if(WNEIsImplemented) {
       result = WaitNextEvent(everyEvent, &Event, Sleep,
         nil);
   }
   else {
       SystemTask();
       result = GetNextEvent(everyEvent, &Event);
   }
   /* Find all application windows in the window list,
and adjust their order if necessary.
   IMPORTANT! Always call LocateWindows() after getting
the event. */
   LocateWindows();

   if(result) {

       switch(Event.what) {

          case mouseDown:

             if(!(ClosingAll || Quitting)) {
                 MouseDownEvent();
             }
             break;

          case keyDown:
          case autoKey:

             if(!(ClosingAll || Quitting)) {
                 KeyEvent();
             }
             break;

          case updateEvt:
             UpdateEvent();
             break;

          case activateEvt:
             ActivateEvent();
             break;

          case app4Evt:
             SuspendResumeEvent();
       }
   }
   else {

       if((ClosingAll || Quitting) && (Event.what
            == nullEvent)) {
```

```
            /* If the application is quitting or the option
         key was held down when closing a window, the
            top visible window is closed during each pass
         through the event loop until all windows are
            closed. */

            if(TopWindow != nil) {
               CloseEvent(TopWindow);
            }
            else {
               ClosingAll = false;

               if(Quitting) {
                  Finished = true;
               }
            }
         }
      }
   }

/*-------------------------MOUSEDOWN EVENT

   AcitiveWindow() replaces most traditional calls to
FrontWindow(). All use of SelectWindow() and DragWindow()
is replaced by calls to DoSelectWindow() and DoDragWin-
dow(). This ensures mouseDown events in application
windows handle floating palettes properly. */

void MouseDownEvent() {
   WindowPtr whichWindow;
   short whichPart;

   MouseInMenu = false;

   whichPart = FindWindow(Event.where, &whichWindow);

   switch(whichPart) {

      case inMenuBar:
         MouseInMenu = true;
         MenuEvent(MenuSelect(Event.where));
         break;

      case inSysWindow:
         SystemClick(&Event, whichWindow);
         break;

      case inContent:
         if(!ActiveWindow(whichWindow)) {
            DoSelectWindow(whichWindow);
         }
         else {
            ContentEvent(whichWindow);
         }
         break;

      case inDrag:
         DoDragWindow(whichWindow);
         break;

      case inGrow:
         if(!ActiveWindow(whichWindow)) {
            DoSelectWindow(whichWindow);
         }
         else {
            GrowEvent(whichWindow);
         }
         break;

      case inGoAway:
         if(!ActiveWindow(whichWindow)) {
            DoSelectWindow(whichWindow);
         }
         else {
            if(TrackGoAway(whichWindow, Event.where)) {
               CloseEvent(whichWindow);
            }
         }
         break;

      case inZoomIn:
      case inZoomOut:
         if(TrackBox(whichWindow, Event.where,
         whichPart)) {
            SetPort(whichWindow);
            EraseRect(&whichWindow->portRect);
            ZoomWindow(whichWindow, whichPart, false);
            SizeDocument(whichWindow);
         }
   }
}

/*-------------------------KEYDOWN AND AUTOKEY EVENT */

void KeyEvent() {
   char whichChar;

   whichChar = Event.message & charCodeMask;

   if((Event.modifiers & cmdKey) != 0) {

      if(Event.what != autoKey) {
         MenuEvent(MenuKey(whichChar));
      }
   }
}

/*-------------------------ACTIVATE EVENT */

void ActivateEvent() {
   WindowPtr whichWindow;

   /* Ensure all application windows are hilited  cor-
rectly. */
   HiliteUserWindows();
   whichWindow = (WindowPtr) Event.message;

   if(((WindowPeek) whichWindow)->refCon
         != DOCUMENT_WINDOW) {
      /* Pass activate event from palette to
      TopDocument. */
      whichWindow = TopDocument;
   }
   if(whichWindow != nil) {
      /* Ensure document window's controls are properly
   visible and hilited. */
      ActivateDocument(whichWindow, Event.modifiers
            & activeFlag);
   }
}

/*-------------------------UPDATE EVENT */

void UpdateEvent() {
   WindowPtr whichWindow;

   whichWindow = (WindowPtr) Event.message;

   SetPort(whichWindow);

   BeginUpdate(whichWindow);

   if(((WindowPeek) whichWindow)->refCon
         == DOCUMENT_WINDOW) {
      UpdateDocument(whichWindow);
   }
   else {
      UpdatePalette(whichWindow);
```

```
    }
    DrawControls(whichWindow);
    DrawGrowIcon(whichWindow);

    EndUpdate(whichWindow);
}

/*-------------------------------SUSPEND/RESUME EVENT

   Hide all palettes when the application is not the active
layer. This leaves the screen much less cluttered. */

void SuspendResumeEvent() {
    short index;

    for(index = 0; index < PALETTE_COUNT; index++) {

        if((Event.message & 1) == 0) {
            PalettesVisible[index] = ((WindowPeek)
            Palettes[index])->visible;
            ShowHide(Palettes[index], false);
        }
        else {
            ShowHide(Palettes[index],
        PalettesVisible[index]);
            PalettesVisible[index] = false;
        }
    }
    LocateWindows();
    HiliteUserWindows();

    if((((WindowPeek) TopWindow)->windowKind == userKind)
        && (TopDocument != nil)) {
        /* Ensure the TopDocument's controls are properly
    visible and unhilited. */
        ActivateDocument(TopDocument, Event.message & 1);
    }
}

/*-------------------------------MENU AND COMMAND KEY EVENT */

void MenuEvent(menuChoice)
long menuChoice;
{
    short whichMenu;
    short whichItem;

    /* HiWord(menuChoice). */
    whichMenu = menuChoice >> 16;
    /* LoWord(menuChoice). */
    whichItem = menuChoice & 0xFFFF;

    switch(whichMenu) {

        case APPLE_MENU_ID:
            AppleMenu(whichItem);
            break;

        case FILE_MENU_ID:
            FileMenu(whichItem);
            break;

        case EDIT_MENU_ID:
            EditMenu(whichItem);
            break;

        case TOOL_MENU_ID:
            ToolMenu(whichItem);
            break;

        case PATTERN_MENU_ID:
            PatternMenu(whichItem);
            break;

        case COLOR_MENU_ID:
            ColorMenu(whichItem);
    }
    HiliteMenu(0);
}

/*-------------------------------SELECT APPLE MENU ITEM */

void AppleMenu(whichItem)
short whichItem;
{
    Str255 daName;

    if(whichItem == ABOUT_ITEM) {
        DoAbout();
    }
    else {
        GetItem(Menus[APPLE_MENU_INDEX], whichItem,
            &daName);
        /* Ignore the result returned by the ROM. */
        (void) OpenDeskAcc(&daName);
    }
}

/*-------------------------------SELECT FILE MENU ITEM */

void FileMenu(whichItem)
short whichItem;
{
    switch(whichItem) {

        case NEW_ITEM:
            NewDocument();
            break;

        case CLOSE_ITEM:
            CloseEvent(TopDocument);
            break;

        case QUIT_ITEM:
            Quitting = true;
            Sleep = 0;
    }
}

/*-------------------------------SELECT EDIT MENU ITEM */

void EditMenu(whichItem)
short whichItem;
{
    (void) SystemEdit(whichItem - 1);
}

/*-------------------------------SELECT TOOL MENU ITEM */

void ToolMenu(whichItem)
short whichItem;
{
    WindowPtr paletteWindow;

    paletteWindow = Palettes[TOOL_PALETTE];

    if(whichItem == MOVE_PALETTE_ITEM) {
        /* The menu has been torn off. */
        MovePalette(paletteWindow,
(*TearOffs[TOOL_PALETTE])->position);
    }
    else {

        if(whichItem
              != (*TearOffs[TOOL_PALETTE])
              ->currentItem) {
            SetPort(Palettes[TOOL_PALETTE]);
```

```
            /* Unhilte the old item. */
            HiliteToolItem(&paletteWindow->portRect,
               (*TearOffs[TOOL_PALETTE])->currentItem,
               false);
            /* Hilite the new item. */
            HiliteToolItem(&paletteWindow->portRect,
               whichItem, true);

            /* Set currentItem equal to the new item. */
            (*TearOffs[TOOL_PALETTE])->currentItem
                  = whichItem;
         }
      }
}

/*-------------------------SELECT PATTERN MENU ITEM */

void PatternMenu(whichItem)
short whichItem;
{
   WindowPtr paletteWindow;

   paletteWindow = Palettes[PATTERN_PALETTE];

   if(whichItem == MOVE_PALETTE_ITEM) {
       /* The menu has been torn off. */
       MovePalette(paletteWindow,
(*TearOffs[PATTERN_PALETTE])->position);
   }
   else {

       if(whichItem
            != (*TearOffs[PATTERN_PALETTE])
            ->currentItem) {
          SetPort(Palettes[PATTERN_PALETTE]);

          /* Unhilte the old item. */
          HilitePatternItem(&paletteWindow->portRect,
             (*TearOffs[PATTERN_PALETTE])
                ->currentItem, false);
          /* Hilite the new item. */
          HilitePatternItem(&paletteWindow->portRect,
             whichItem, true);

          /* Set currentItem equal to the new item. */
          (*TearOffs[PATTERN_PALETTE])->currentItem =
             whichItem;
       }
   }
}

/*-------------------------SELECT COLOR MENU ITEM */

void ColorMenu(whichItem)
short whichItem;
{
   WindowPtr paletteWindow;

   paletteWindow = Palettes[COLOR_PALETTE];

   if(whichItem == MOVE_PALETTE_ITEM) {
       /* The menu has been torn off. */
       MovePalette(paletteWindow,
(*TearOffs[COLOR_PALETTE])->position);
   }
   else {

       if(whichItem
            != (*TearOffs[COLOR_PALETTE])
            ->currentItem) {
          SetPort(Palettes[COLOR_PALETTE]);

          /* Unhilte the old item. */
          HiliteColorItem(&paletteWindow->portRect,
             (*TearOffs[COLOR_PALETTE])->currentItem,
             false);
          /* Hilite the new item. */
          HiliteColorItem(&paletteWindow->portRect,
             whichItem, true);

          /* Set currentItem equal to the new item. */
          (*TearOffs[COLOR_PALETTE])->currentItem =
             whichItem;
       }
   }
}

/*-------------------------CONTENT EVENT */

void ContentEvent(whichWindow)
WindowPtr whichWindow;
{
   Point mousePt;
   short item;

   SetPort(whichWindow);

   mousePt = Event.where;
   GlobalToLocal(&mousePt);

   /* Use the window's refCon to determine what type of
application window received the event. */
   switch(((WindowPeek) whichWindow)->refCon) {

       case TOOL_PALETTE:
          item = FindToolItem(mousePt);

          if(item
                != (*TearOffs[TOOL_PALETTE])
                ->currentItem) {
             /* Unhilte the old item. */
             HiliteToolItem(&whichWindow->portRect,
                (*TearOffs[TOOL_PALETTE])->
                   currentItem, false);
             /* Hilite the new item. */
             HiliteToolItem(&whichWindow->portRect, item,
                true);

             /* Set currentItem equal to the new item. */
             (*TearOffs[TOOL_PALETTE])->currentItem
                   = item;
          }
          break;

       case PATTERN_PALETTE:
          item = FindPatternItem(mousePt);

          if(item
                != (*TearOffs[PATTERN_PALETTE])
                ->currentItem) {
             /* Unhilte the old item. */
             HilitePatternItem(&whichWindow->portRect,
                (*TearOffs[PATTERN_PALETTE])
                   ->currentItem, false);
             /* Hilite the new item. */
             HilitePatternItem(&whichWindow->portRect,
                item, true);

             /* Set currentItem equal to the new item. */
             (*TearOffs[PATTERN_PALETTE])->currentItem
                   = item;
          }
          break;

       case COLOR_PALETTE:
          item = FindColorItem(mousePt);

          if(item
```


© The Definitive MacTutor, Vol. 4

```
                != (*TearOffs[COLOR_PALETTE])
                ->currentItem) {
            /* Unhilte the old item. */
            HiliteColorItem(&whichWindow->portRect,
                (*TearOffs[COLOR_PALETTE])
                    ->currentItem, false);
            /* Hilite the new item. */
            HiliteColorItem(&whichWindow->portRect,
                item, true);

            /* Set currentItem equal to the new item. */
            (*TearOffs[COLOR_PALETTE])->currentItem
                    = item;
        }
    }
}

/*-----------------------------GROW EVENT */

void GrowEvent(whichWindow)
WindowPtr whichWindow;
{
    Rect growRect;
    long size;
    short width;
    short height;

    growRect.left = MIN_WINDOW_WIDTH;
    growRect.top = MIN_WINDOW_HEIGHT;
    growRect.right = (*GrayRgn)->rgnBBox.right;
    growRect.bottom = (*GrayRgn)->rgnBBox.bottom
        - TITLE_BAR_HEIGHT;

    size = GrowWindow(whichWindow, Event.where,
    &growRect);

    if(size != 0) {
        SetPort(whichWindow);

        InvalGrow(whichWindow);

        height = size >> 16; /* HiWord(size). */
        width = size & 0xFFFF; /* LoWord(size). */

        SizeWindow(whichWindow, width, height, true);
        /* Redraw the scroll bars and grow icon. */
        SizeDocument(whichWindow);
    }
}

/*-----------------------------INVAL GROW

    Invalidate the grow region of the window. This includes
the area occupied by the scroll bars. */

void InvalGrow(whichWindow)
WindowPtr whichWindow;
{
    Rect updateRect;

    updateRect = whichWindow->portRect;

    updateRect.top = updateRect.bottom - SCROLL_BAR_SIZE;
    InvalRect(&updateRect);

    updateRect.top = whichWindow->portRect.top;

    updateRect.left = updateRect.right - SCROLL_BAR_SIZE;
    InvalRect(&updateRect);
}

/*-----------------------------CLOSE EVENT */

void CloseEvent(whichWindow)
WindowPtr whichWindow;
{
    WindowPeek front;

    if(Event.modifiers & optionKey) {
        ClosingAll = true;
    }
    if ((front = (WindowPeek) TopWindow) != nil) {

        if(front->windowKind < 0) {
            CloseDeskAcc(front->windowKind);
        }
        else {

            if((front->windowKind == userKind)
                    && (whichWindow != nil)) {

                if(((WindowPeek) whichWindow)->refCon
                        == DOCUMENT_WINDOW) {
                    CloseDocument(whichWindow);

                    if(TopPalette == TopWindow) {
                        /* Find the new TopDocument. */
                        LocateWindows();

                        if(TopDocument != nil) {
                            /* Ensure the TopDocument and its
                        controls are hilited properly. */
                            ActivateDocument(TopDocument,
                                true);
                        }
                    }
                }
                else {

                    if(TopDocument != nil) {
                        /* Don't generate any events that
                    might cause the TopDocument to be
                    unhilited. */
                        ShowHide(whichWindow, false);
                    }
                    else {
                        /* Ensure an activate event is
                    generated in case the next visible
                    window does not belong to the
                application. */
                        HideWindow(whichWindow);
                    }
                }
            }
        }
    }
}

/*-------TEAROFF PALETTE

    version 1.0
    by Don Melton and Mike Ritter

    Copyright (C)1987, 1988 by Impulse Technologies,
    Inc., all rights reserved.

    Filename:       Window.c
    Font:           Monaco, 9 point
    Tab setting:    2
    Compiler:       LightspeedC 2.15
    Project type: APPL
    Creator:        TOPD
    Segment:        Main */

/*GLOBAL CONSTANT DEFINITIONS AND VARIABLE DECLARATIONS */

#include "Constants.h"
#include "Variables.h"
```

```
/*------------------------EXTERNAL FUNCTION DECLARATIONS */

extern short GoodNewPtr();              /* Error.c */
extern short GoodResource();            /* Error.c */

extern short GetMenuBarHeight();        /* Interface.c */

extern pascal void DrawToolMenu();      /* Menu.c */
extern pascal void HiliteToolItem();    /* Menu.c */
extern pascal void DrawPatternMenu();   /* Menu.c */
extern pascal void HilitePatternItem(); /* Menu.c */
extern pascal void DrawColorMenu();     /* Menu.c */
extern pascal void HiliteColorItem();   /* Menu.c */

/*------------------------FORWARD FUNCTION DECLARATIONS */

void NewDocument();
Rect *GetDocumentRect();

short NewScrollBars();
Rect *GetHScrollBarRect();
Rect *GetVScrollBarRect();

void CloseDocument();
void ActivateDocument();
void UpdateDocument();
void SizeDocument();

void MoveSizeScrollBar();

void UpdatePalette();

/*------------------------NEW DOCUMENT */

void NewDocument() {
    StringHandle title;
    WindowPtr theDocument;
    Rect windowRect;

    if(!GoodResource(title =
GetString(DOCUMENT_TITLE_STRING_ID))) {
        return;
    }
    HLock(title);

    if(!GoodNewPtr(theDocument = NewWindow(nil,
          GetDocumentRect(&windowRect),
          *title,
          NOT_VISIBLE,
          zoomDocProc,
          BottomPalette,
          GO_AWAY_BOX,
          DOCUMENT_WINDOW))) {
       HUnlock(title);

       return;
    }
    if(!NewScrollBars(theDocument)) {
       CloseDocument(theDocument);
       PositionCounter += 1;
       HUnlock(title);

       return;
    }
    PositionCounter += 1;
    HUnlock(title);

    if(TopPalette != nil) {

       if(TopPalette != TopWindow) {
          /* Bring the TopPalette to the front and
    generate activate event. */
          SelectWindow(TopPalette);
       }
       else {
          HiliteWindow(theDocument, true);
       }
       if(TopDocument != nil) {
          /* Ensure the TopDocument and its controls are
       unhilited properly. */
          ActivateDocument(TopDocument, false);
       }
    }
    ShowWindow(theDocument);
}

/*------------------------GET DOCUMENT RECT */

Rect *GetDocumentRect(windowRect)
Rect *windowRect;
{
    short position;

    position = (PositionCounter + DOCUMENT_COUNT)
          % DOCUMENT_COUNT; windowRect->left
          = SCREEN_MARGIN
          + (HORIZONTAL_WINDOW_OFFSET * position);
          windowRect->top = GetMenuBarHeight()
          + TITLE_BAR_HEIGHT + SCREEN_MARGIN
          + (VERTICAL_WINDOW_OFFSET * position);
          windowRect->right = screenBits.bounds.right
          - SCREEN_MARGIN - (((DOCUMENT_COUNT - 1)
          - position) * HORIZONTAL_WINDOW_OFFSET);
          windowRect->bottom = screenBits.bounds.bottom
          - SCREEN_MARGIN - (((DOCUMENT_COUNT - 1)
          - position) * VERTICAL_WINDOW_OFFSET);
    return(windowRect);
}

/*------------------------NEW SCROLL BARS */

short NewScrollBars(whichWindow)
WindowPtr whichWindow;
{
    Rect sBRect;

    if(GoodNewHandle(NewControl(whichWindow,
          GetHScrollBarRect(whichWindow, &sBRect),
          NULL,
          VISIBLE,
          0, 0, 255,
          scrollBarProc,
          NO_REFCON))) {

       if(GoodNewHandle(NewControl(whichWindow,
          GetVScrollBarRect(whichWindow, &sBRect),
             NULL,
             VISIBLE,
             0, 0, 255,
             scrollBarProc,
             NO_REFCON))) {
          return(true);
       }
    }
    return(false);
}

/*------------------GET HORIZONTAL SCROLL BAR RECTANGLE */

Rect *GetHScrollBarRect(whichWindow, whichRect)
WindowPtr whichWindow;
Rect *whichRect;
{
    *whichRect = whichWindow->portRect;

    whichRect->left -= 1;
    whichRect->top = whichRect->bottom
```

```
        - (SCROLL_BAR_SIZE - 1);
    whichRect->right -= (SCROLL_BAR_SIZE - 2);
    whichRect->bottom += 1;

    return(whichRect);
}

/*-----------------GET VERTICAL SCROLL BAR RECTANGLE */

Rect *GetVScrollBarRect(whichWindow, whichRect)
WindowPtr whichWindow;
Rect *whichRect;
{
    *whichRect = whichWindow->portRect;

    whichRect->left = whichRect->right
        - (SCROLL_BAR_SIZE - 1);
    whichRect->top -= 1;
    whichRect->right += 1;
    whichRect->bottom -= (SCROLL_BAR_SIZE - 2);

    return(whichRect);
}

/*-----------------------------CLOSE DOCUMENT

    A REAL application would need a more complex CloseDocu-
ment() function. This is just a shell. */

void CloseDocument(whichWindow)
WindowPtr whichWindow;
{
    DisposeWindow(whichWindow);

    PositionCounter -= 1;
}

/*-----------------------------ACTIVATE DOCUMENT */

void ActivateDocument(whichWindow, activate)
WindowPtr whichWindow;
Boolean activate;
{
    HiliteWindow(whichWindow, activate);

    if(activate) {
        ShowControl(((WindowPeek)
whichWindow)->controlList);
        ShowControl((*((WindowPeek)
whichWindow)->controlList)->nextControl);
    }
    else {
        SetPort(whichWindow);
        HideControl(((WindowPeek)
whichWindow)->controlList);
        ValidRect(&(*((WindowPeek)
whichWindow)->controlList)->contrlRect);
        HideControl((*((WindowPeek)
whichWindow)->controlList)->nextControl);
        ValidRect(&(*(*((WindowPeek)
    whichWindow)->controlList)
            ->nextControl)->contrlRect);
    }
    DrawGrowIcon(whichWindow);
}

/*-----------------------------UPDATE DOCUMENT

    A REAL application would need a more complex UpdateDocu-
ment() function. This is just a shell. */

void UpdateDocument(whichDocument)
WindowPtr whichDocument;
{
    EraseRect(&whichDocument->portRect);
}

/*-----------------------------SIZE DOCUMENT */

void SizeDocument(whichDocument)
WindowPtr whichDocument;
{
    Rect sBRect;

    MoveSizeScrollBar(((WindowPeek)
whichDocument)->controlList,
        GetHScrollBarRect(whichDocument, &sBRect));
            MoveSizeScrollBar((*((WindowPeek)
            whichDocument)->controlList)
                ->nextControl,
        GetVScrollBarRect(whichDocument, &sBRect));

    BeginUpdate(whichDocument);
    UpdateDocument(whichDocument);
    EndUpdate(whichDocument);

    ShowControl(((WindowPeek)whichDocument)->controlList);
    ShowControl((*((WindowPeek)whichDocument)->controlList)-
>nextControl);

    DrawGrowIcon(whichDocument);
}

/*-----------------------------MOVE AND SIZE SCROLL BAR */

void MoveSizeScrollBar(whichScrollBar, sBRect)
ControlHandle whichScrollBar;
Rect *sBRect;
{
    HideControl(whichScrollBar);

    MoveControl(whichScrollBar, sBRect->left,
    sBRect->top);
    SizeControl(whichScrollBar, sBRect->right -
    sBRect->left, sBRect->bottom - sBRect->top);
}

/*-----------------------------UPDATE PALETTE */

void UpdatePalette(whichPalette)
WindowPtr whichPalette;
{
    /* Use the palette's refCon to determine which  palette
needs updating. */
    switch(((WindowPeek) whichPalette)->refCon) {

        case TOOL_PALETTE:
            DrawToolMenu(&whichPalette->portRect);
            HiliteToolItem(&whichPalette->portRect,
            (*TearOffs[TOOL_PALETTE])->currentItem,
            true);
            break;

        case PATTERN_PALETTE:
            DrawPatternMenu(&whichPalette->portRect);
            HilitePatternItem(&whichPalette->portRect,
                (*TearOffs[PATTERN_PALETTE])
                    ->currentItem, true);
            break;

        case COLOR_PALETTE:
            DrawColorMenu(&whichPalette->portRect);
            HiliteColorItem(&whichPalette->portRect,
            (*TearOffs[COLOR_PALETTE])->currentItem,
            true);
    }
}
```

```
/*-------------------------TEAROFF PALETTE

   version 1.0
   by Don Melton and Mike Ritter

   Copyright (C)1987, 1988 by Impulse Technologies,
   Inc., all rights reserved.

   Filename:        Menu.c
   Font:            Courier, 9 point
   Tab setting:     2
   Compiler:        LightspeedC 2.15
   Project type: APPL
   Creator:         TOPD
   Segment:         Main */

/*GLOBAL CONSTANT DEFINITIONS AND VARIABLE DECLARATIONS */

#include "Constants.h"
#include "Variables.h"

/*-------------------------EXTERNAL FUNCTION DECLARATIONS */

/* None */

/*-------------------------FORWARD FUNCTION DECLARATIONS */

pascal void DrawToolMenu();
pascal short FindToolItem();
pascal void HiliteToolItem();

pascal void DrawPatternMenu();
pascal short FindPatternItem();
pascal void HilitePatternItem();

pascal void DrawColorMenu();
pascal short FindColorItem();
pascal void HiliteColorItem();

void MovePalette();



/*-------------------------DRAW TOOLS */

pascal void DrawToolMenu(destRect)
Rect *destRect;
{
   DrawPicture(GetPicture(TOOL_PICT_ID), destRect);
}

/*-------------DRAW PATTERNS */

pascal void DrawPatternMenu(destRect)
Rect *destRect;
{
    short index;
    Rect drawRect;
    Pattern thePattern;

    if(MouseInMenu) {
       EraseRect(destRect);
    }
    for(index = 1; index < PATTERN_COUNT; index++) {
       drawRect = PatternRects[index];
       drawRect.top += 1;
       drawRect.left += 1;
       OffsetRect(&drawRect, destRect->left,
       destRect->top);

       GetIndPattern(&thePattern, PATTERNS_ID, index);
       FillRect(&drawRect, &thePattern);
    }
}


/*-------------DRAW COLORS */

pascal void DrawColorMenu(destRect)
Rect *destRect;
{
   CTabPtr colorTable;
   RGBColor saveColor;
   short index;
   Rect drawRect;

   if(MouseInMenu) {
      EraseRect(destRect);
   }
   HLock((*(*MaxDevice)->gdPMap)->pmTable);
   colorTable = *(*(*MaxDevice)->gdPMap)->pmTable;

   GetForeColor(&saveColor);

   for(index = 1; index < COLOR_COUNT; index++) {
      RGBForeColor(&colorTable->ctTable[index - 1].rgb);
      drawRect = ColorRects[index];
      drawRect.top += 1;
      drawRect.left += 1;
      OffsetRect(&drawRect, destRect->left,
            destRect->top);

      /* Remember black is a Color QuickDraw pattern. */
      FillRect(&drawRect, black);
   }
   RGBForeColor(&saveColor);

   HUnlock((*(*MaxDevice)->gdPMap)->pmTable);
}

/*-------------------------FIND TOOL ITEM */

pascal short FindToolItem(mousePt)
Point mousePt;
{
   short index;

   for(index = 1; index < TOOL_COUNT; index++) {

      if(PtInRect(mousePt, &ToolRects[index])) {
         return(index);
      }
   }
   return(0);
}

/*-------------FIND PATTERN ITEM */

pascal short FindPatternItem(mousePt)
Point mousePt;
{
    short index;
    Rect patternRect;

    for(index = 1; index < PATTERN_COUNT; index++) {

       patternRect = PatternRects[index];
       patternRect.bottom += 1;
       patternRect.right += 1;

       if(PtInRect(mousePt, &patternRect)) {
          return(index);
       }
    }
    return(0);
}

/*-------------FIND COLOR ITEM */
```

```
pascal short FindColorItem(mousePt)
Point mousePt;
{
    short index;
    Rect colorRect;

    for(index = 1; index < COLOR_COUNT; index++) {

        colorRect = ColorRects[index];
        colorRect.bottom += 1;
        colorRect.right += 1;

        if(PtInRect(mousePt, &colorRect)) {
            return(index);
        }
    }
    return(0);
}

/*-------------HILITE TOOL ITEM */

pascal void HiliteToolItem(destRect, item, hilite)
Rect *destRect;
short item;
Boolean hilite;
{
    Rect hiliteRect;

    hiliteRect = ToolRects[item];
    hiliteRect.bottom -= 1;
    hiliteRect.right -= 1;
    OffsetRect(&hiliteRect, destRect->left,
destRect->top);
    InvertRect(&hiliteRect);
}

/*-------------HILITE PATTERN ITEM */

pascal void HilitePatternItem(destRect, item, hilite)
Rect *destRect;
short item;
Boolean hilite;
{
    Rect hiliteRect;
    PenState savePen;
    Pattern thePattern;

    hiliteRect = PatternRects[item];
    hiliteRect.top += 1;
    hiliteRect.left += 1;
    OffsetRect(&hiliteRect, destRect->left,
destRect->top);

    if(hilite) {
        GetPenState(&savePen);

        PenSize(2, 2);
        FrameRect(&hiliteRect);

        PenSize(1, 1);
        PenMode(patBic);
        InsetRect(&hiliteRect, 2, 2);
        FrameRect(&hiliteRect);

        SetPenState(&savePen);
    }
    else {
        GetIndPattern(&thePattern, PATTERNS_ID, item);
        FillRect(&hiliteRect, &thePattern);
    }
}

/*-------------------HILITE COLOR ITEM */

pascal void HiliteColorItem(destRect, item, hilite)
Rect *destRect;
short item;
Boolean hilite;
{
    PenState savePen;
    Rect hiliteRect;
    RGBColor saveColor;
    CTabPtr colorTable;

    GetPenState(&savePen);

    hiliteRect = ColorRects[item];
    hiliteRect.bottom += 1;
    hiliteRect.right += 1;
    OffsetRect(&hiliteRect, destRect->left,
destRect->top);

    if(hilite) {
        FrameRect(&hiliteRect);

        PenMode(patBic);
        InsetRect(&hiliteRect, 1, 1);
        FrameRect(&hiliteRect);

        SetPenState(&savePen);
    }
    else {
        GetForeColor(&saveColor);

        HLock((*(*MaxDevice)->gdPMap)->pmTable);
        colorTable = *(*(*MaxDevice)->gdPMap)->pmTable;

        PenMode(patBic);
        FrameRect(&hiliteRect);

        SetPenState(&savePen);
        RGBForeColor(&colorTable->ctTable[item - 1].rgb);

        InsetRect(&hiliteRect, 1, 1);
        /* Remember black is a Color QuickDraw pattern. */
        FillRect(&hiliteRect, black);

        HUnlock((*(*MaxDevice)->gdPMap)->pmTable);

        RGBForeColor(&saveColor);
    }
}

/*-------------------------------MOVE PALETTE */

void MovePalette(whichPalette, position)
WindowPtr whichPalette;
Point position;
{
    MoveWindow(whichPalette, position.h, position.v,
    false);

    if((TopWindow != nil) && (((WindowPeek)
TopWindow)->windowKind != userKind)) {

        /* The TopWindow is a non-application document. */
        if(((WindowPeek) whichPalette)->visible) {
            /* The palette is visible. Bring it to the
        front and generate an activate event. */
            SelectWindow(whichPalette);
        }
        else {
            BringToFront(whichPalette);
            /* Make the palette visible and generate an
        activate event. */
            ShowWindow(whichPalette);
        }
    }
```

```
    else {
       /* The TopWindow is an application window or it is
    equal to nil. */
       BringToFront(whichPalette);

       if(TopWindow == nil) {
          /* Make the palette visible and generate an
       activate event. */
          ShowWindow(whichPalette);
       }
       else {
          /* Make the palette visible but don't generate
       an activate event. */
          ShowHide(whichPalette, true);
          HiliteWindow(whichPalette, true);
       }
    }
}

/*-------------------------TEAROFF  PALETTE

    version 1.0
    by Don Melton and Mike Ritter

    Copyright (C)1987, 1988 by Impulse Technologies,
    Inc., all rights reserved.

    Filename:       Interface.c
    Font:              Courier, 9 point
    Tab setting:    2
    Compiler:       LightspeedC 2.15
    Project type:   APPL
    Creator:        TOPD
    Segment:           Main */

/*GLOBAL CONSTANT DEFINITIONS AND VARIABLE DECLARATIONS */

#include "Constants.h"
#include "Variables.h"

/*-----------------------EXTERNAL FUNCTION DECLARATIONS */

/* None */

/*-----------------------FORWARD FUNCTION DECLARATIONS */

void AdjustInterface();
short GetMenuBarHeight();

/*-------------------------ADJUST INTERFACE */

void AdjustInterface() {

    if(Environment.hasColorQD) {
       MaxDevice = GetMaxDevice(&(*GrayRgn)->rgnBBox);

       if((*(*MaxDevice)->gdPMap)->pixelSize == 8) {

          if(!ColorMenuVisible) {
             /* Insert Color menu if 256 colors. */
             InsertMenu(Menus[COLOR_MENU_INDEX], 0);
             DrawMenuBar();
             ColorMenuVisible = true;
          }
       }
       else {

          if(ColorMenuVisible) {
             /* Remove Color menu if < 256 colors. */
             DeleteMenu(COLOR_MENU_ID);
             DrawMenuBar();
             ColorMenuVisible = false;

             if(((WindowPeek)
                Palettes[COLOR_PALETTE])->visible) {
                   HideWindow(Palettes[COLOR_PALETTE]);
                }
                PalettesVisible[COLOR_PALETTE] = false;
             }
          }
       }
    }
}

/*-------------------------GET MENU BAR HEIGHT */

short GetMenuBarHeight() {

    if(Environment.machineType < envMachUnknown) {
       /* Arrggh! Someone has the 64K ROMs. */
       return(20);
    }
    else {
       return(MBarHeight);
    }
}

/*-------------------------TEAROFF  PALETTE

    version 1.0
    by Don Melton and Mike Ritter

    Copyright (C)1987, 1988 by Impulse Technologies,
    Inc., all rights reserved.

    Filename:       Palette.c
    Font:           Courier, 9 point
    Tab setting:    2
    Compiler:       LightspeedC 2.15
    Project type: APPL
    Creator:        TOPD
    Segment:        Main */

/*GLOBAL CONSTANT DEFINITIONS AND VARIABLE DECLARATIONS */

#include "Constants.h"
#include "Variables.h"

/*-----------------------EXTERNAL FUNCTION DECLARATIONS */

extern void ActivateDocument();    /* Window.c */

extern short GetMenuBarHeight();   /* Interface.c */

/*-------------------------FORWARD FUNCTION DECLARATIONS */

void LocateWindows();
void HiliteUserWindows();
short ActiveWindow();

void DoDragWindow();
void DoSelectWindow();
void SendToBack();
void Drag();

/*-------------------------LOCATE WINDOWS

   Find the TopWindow whether it belongs to the application
or not. Also find the TopPalette, the BottomPalette, and the
TopDocument. These globals will all point to visible windows
or nil. If the TopDocument is above any visible palette,
adjust the order of the windows. If a visible non-applica-
tion window is between any palettes and the TopDocument,
adjust the order of  the windows. This function replaces
most uses of FrontWindow() in the application. */

void LocateWindows()
{
   short palettesFound;
```

```
    Boolean inOrder;
    WindowPeek next;
    WindowPeek bottom;

    /* Initialize globals before search is begun. */
    TopWindow = TopPalette = TopDocument = nil;
    /* Make sure all SendBehind() and NewWindow() calls
bring a window to the front if there are no visible palettes.
*/
    BottomPalette = BRING_TO_FRONT;

    palettesFound = 0;
    inOrder = true;

    for(next = WindowList; next != nil;
          next = next->nextWindow) {

       if(next->visible) {

          if(TopWindow == nil) {
             /* First visible window. */
             TopWindow = (WindowPtr) next;
          }
          if(next->windowKind == userKind) {

             if(next->refCon == DOCUMENT_WINDOW) {

                if(TopDocument == nil) {
                   /* First visible Document. */
                   TopDocument = (WindowPtr) next;

                   if(palettesFound == PALETTE_COUNT) {
                      /* If all the palettes are found
                   exit the search loop. */
                      break;
                   }
                }
             }
             else {

                if(TopPalette == nil) {
                   /* First visible palette. */
                   TopPalette = (WindowPtr) next;
                }
                /* BottomPalette always points to the
             last visible palette found. */
                BottomPalette = (WindowPtr) next;
             }
          }
          else {

             if((TopPalette != nil)
                   || (TopDocument != nil)) {
                /* Shucks, a non-application window is
             visible between the palettes and
          TopDocument. */
                inOrder = false;
             }
          }
       }
       /* All windows beforelast palette get here */
       if((next->windowKind == userKind) && (next->refCon
          != DOCUMENT_WINDOW)) {
          palettesFound++;

          if((TopDocument != nil) && (next->visible)) {
             /* A visible palette is beneath the
          TopDocument. */

             if(TopDocument == TopWindow) {
                /* Bring the TopPalette to the front. */
                SelectWindow(next);
                TopWindow = (WindowPtr) next;
             }
             else {
                /* Put TopDocument behind BottomPalette. */
                SendBehind(TopDocument, next);
             }
             next = (WindowPeek) TopDocument;

             if(palettesFound == PALETTE_COUNT) {
                /* If all the palettes are found exit the
             search loop. */
                break;
             }
          }
       }
    }
    if((!inOrder) && (TopPalette != nil)) {

       if(TopDocument != nil) {
          bottom = (WindowPeek) TopDocument;
       }
       else {
          bottom = (WindowPeek) BottomPalette;
       }
       for(next = (WindowPeek) TopPalette; next != nil;
          next = next->nextWindow) {

          if(next != bottom) {

             if(next->windowKind != userKind) {
                /* Remove the non-application window. */
                SendBehind(next, bottom);
                /* Start over. */
                next = (WindowPeek) TopPalette;
             }
          }
          else {
             break;
          }
       }
    }
}

/*-----------------------------HILITE USER WINDOWS */

void HiliteUserWindows()
{
    WindowPeek bottom;
    WindowPeek next;
    Boolean hilite;
    short index;

    if(TopPalette != nil) {

       if(TopWindow == TopPalette) {
          hilite = true;
       }
       else {
          hilite = false;
       }
       for(index = 0; index < PALETTE_COUNT; index++) {
          HiliteWindow(Palettes[index], hilite);
       }
       if(TopDocument != nil) {

          HiliteWindow(TopDocument, hilite);
          /* Unhilite the remaining document windows. */
          for(next = ((WindowPeek)
       TopDocument)->nextWindow; next != nil;
          next = next->nextWindow) {

             if((next->windowKind == userKind) &&
             (next->refCon == DOCUMENT_WINDOW)) {
                ActivateDocument(next, false);
             }
          }
```

```
      }
   }
}

/*-----------------------------ACTIVE WINDOW

   Return true if whichWindow is a palette or the TopDocu-
ment and the application is active. */

short ActiveWindow(whichWindow)
WindowPtr whichWindow;
{
   if(whichWindow != nil) {

      if((((WindowPeek) TopWindow)->windowKind
            == userKind)
            && ((whichWindow == TopDocument)
            || (((WindowPeek) whichWindow)->refCon
            != DOCUMENT_WINDOW))) {
         return(true);
      }
   }
   return(false);
}

/*-----------------------DO DRAG WINDOW

   Since DragWindow() automatically brings a window to the
front if it's not already the front window, it must be
replaced by the application to keep palettes floating above
documents. */

void DoDragWindow(whichWindow)
WindowPtr whichWindow;
{
   if(Event.modifiers & optionKey) {
      /*An optional extension of Macintosh interface. */
      SendToBack(whichWindow);
   }
   else {

      if(Event.modifiers & cmdKey) {
         Drag(whichWindow);
      }
      else {

         if(!ActiveWindow(whichWindow)) {
            DoSelectWindow(whichWindow);
         }
         else {

            if((((WindowPeek) whichWindow)->refCon
                  != DOCUMENT_WINDOW)
                  && (whichWindow != TopPalette)) {
               BringToFront(whichWindow);
               /* Force the palette to be updated before
            it is dragged. */
               SetPort(whichWindow);
               UpdatePalette(whichWindow);
            }
            Drag(whichWindow);
         }
      }
   }
}

/*-----------------------------DO SELECT WINDOW

   Since SelectWindow() brings any window to the front, it
must be replaced by the application to keep palettes
floating above documents. */

void DoSelectWindow(whichWindow)
WindowPtr whichWindow;
{
   RgnHandle updateRgn;

   if(((WindowPeek) whichWindow)->refCon
         == DOCUMENT_WINDOW) {

      if(TopPalette != nil) {
         /* Calculate window area not visible that will
      need to be updated. */
         CopyRgn(whichWindow->visRgn, updateRgn
               = NewRgn());
         OffsetRgn(updateRgn,(*((WindowPeek)
         whichWindow)->contRgn)->rgnBBox.left,
               (*((WindowPeek)whichWindow)
               ->contRgn)->rgnBBox.top);
         DiffRgn(((WindowPeek) whichWindow)->strucRgn,
            updateRgn, updateRgn);

         /* Move document below the BottomPalette. */
         SendBehind(whichWindow, BottomPalette);
         CalcVisBehind(whichWindow, updateRgn);
         PaintOne(whichWindow, updateRgn);

         DisposeRgn(updateRgn);

         if(TopPalette != TopWindow) {
            /* Bring all active application windows
         forward by generating an activate event. */
            SelectWindow(TopPalette);
         }
         else {
            /* Ensure whichWindow and its controls are
         hilited properly. */
            ActivateDocument(whichWindow, true);
         }
         /* Ensure whichWindow and its controls are
      unhilited properly. */
         ActivateDocument(TopDocument, false);
      }
      else {
         /* No Palettes, bring the document to the front
      and generate an activate event. */
         SelectWindow(whichWindow);
      }
   }
   else {
      if((((WindowPeek) TopWindow)->windowKind
            == userKind) {
         /* Bring the palette to the front but don't
      generate an activate event. */
         BringToFront(whichWindow);
      }
      else {
         /* Bring the palette to the front and generate
      an activate event. */
         SelectWindow(whichWindow);
      }
   }
}

/*-----------------------------SEND TO BACK */

void SendToBack(whichWindow)
WindowPtr whichWindow;
{
   WindowPeek next;
   WindowPtr below;

   if(((WindowPeek) whichWindow)->refCon
         == DOCUMENT_WINDOW) {

      below = whichWindow;

      /* Find the document below TopDocument. Documents
```

```
    are always visible. */
       for(next = ((WindowPeek) whichWindow)->nextWindow;
          next != nil; next = next->nextWindow) {

          if((next->windowKind == userKind) &&
          (next->refCon == DOCUMENT_WINDOW)) {
             below = (WindowPtr) next;
             break;
          }
       }
       if(whichWindow != below) {
          /* Send document all the way to the back. */
          SendBehind(whichWindow, nil);

          if(TopPalette == TopWindow) {
             /* Ensure whichWindow and its controls are
          unhilited properly. */
             ActivateDocument(whichWindow, false);

             /* Find the new TopDocument. */
             LocateWindows();
             /* Ensure the TopDocument and its controls
          are hilited properly. */
             ActivateDocument(TopDocument, true);
          }
       }
    }
    else {

       if((whichWindow != BottomPalette) && (TopPalette
          != BottomPalette)) {
          /* Move palette behind the BottomPalette. */
          SendBehind(whichWindow, BottomPalette);
       }
    }
}

/*-------------------------------DRAG */

void Drag(whichWindow)
WindowPtr whichWindow;
{
    GrafPtr savePort;
    Rect dragLimit;
    RgnHandle dragRgn;
    long result;
    Point move;
    Rect *portBounds;

    if(WaitMouseUp()) {
       GetPort(&savePort);
       SetPort(WMgrPort);

       SetClip(GrayRgn);

       /* Ensure the drag outline does not get drawn
    through the windows above whichWindow. */
       ClipAbove(whichWindow);

       dragLimit = screenBits.bounds;
       dragLimit.top = GetMenuBarHeight();

       CopyRgn(((WindowPeek) whichWindow)->strucRgn,
          dragRgn = NewRgn());

       /* Drag outline of window around the screen. */
       result = DragGrayRgn(dragRgn, Event.where,
       &dragLimit, &dragLimit, noConstraint, nil);

       move.v = result >> 16; /* HiWord(result). */
       move.h = result & 0xFFFF; /* LoWord(result). */

       if(move.v != 0x8000) {
          /* The mouse button has been released inside
       dragLimit. See 'Inside Macintosh Volume I,'
    pages 294-295, for expalnation of 0x8000. */
          if(((CGrafPtr) whichWindow)->portVersion
                & 0xc000) {
             /* The window is a color port. */
             portBounds = &(*((CGrafPtr)
          whichWindow)->portPixMap)->bounds;
          }
          else {
             /* The window is an old-style port. */
             portBounds = &whichWindow->portBits.bounds;
          }
          move.v += whichWindow->portRect.top
                - portBounds->top;
          move.h += whichWindow->portRect.left
                - portBounds->left;

          MoveWindow(whichWindow, move.h, move.v, false);
       }
       DisposeRgn(dragRgn);
       SetPort(savePort);
    }
}

/*-------------------------TEAROFF  PALETTE

    version 1.0
    by Don Melton and Mike Ritter

    Copyright (C)1987, 1988 by Impulse Technologies,
        Inc., all rights reserved.

    Filename:           Dialog.c
    Font:               Courier, 9 point
    Tab setting:        2
    Compiler:           LightspeedC 2.15
    Project type:   APPL
    Creator:        TOPD
    Segment:        Main */

/*GLOBAL CONSTANT DEFINITIONS AND VARIABLE DECLARATIONS */

#include "Constants.h"
#include "Variables.h"

/*--------------------EXTERNAL FUNCTION DECLARATIONS */

extern short GoodResource();        /* Error.c */
extern short GetMenuBarHeight();    /* Interface.c */

/*--------------------FORWARD FUNCTION DECLARATIONS */

void DoAbout();
short PositionDialog();
pascal Boolean DialogFilter();

/*-------------------------------DO ABOUT */

void DoAbout() {
    DialogRecord theDialogRecord;
    short itemHit;

    if(PositionDialog(ABOUT_DIALOG_ID)) {
       (void) GetNewDialog(ABOUT_DIALOG_ID,
    &theDialogRecord, BRING_TO_FRONT);
       ShowWindow(&theDialogRecord);

       do {
          ModalDialog(DialogFilter, &itemHit);
       }
       while(itemHit == 0);

       CloseDialog(&theDialogRecord);
```

```
    }
}

/*-----------------------------POSITION DIALOG

    PositionDialog() centers a DITL resource horizontally
and positions it one-third down
the screen vertically. */

short PositionDialog(whichID)
short whichID;
{
    Point where;
    DialogTHndl theTemplate;

    if(!GoodResource(theTemplate = (DialogTHndl)
    GetResource('DLOG', whichID))) {
        return(false);
    }
    OffsetRect(&(*theTemplate)->boundsRect,
            - (*theTemplate)->boundsRect.left
            + ((screenBits.bounds.right
            - screenBits.bounds.left)
            - ((*theTemplate)->boundsRect.right
            - (*theTemplate)->boundsRect.left)) / 2,
            - (*theTemplate)->boundsRect.top
            + (((screenBits.bounds.bottom
            - screenBits.bounds.top-GetMenuBarHeight())
            - ((*theTemplate)->boundsRect.bottom
            - (*theTemplate)->boundsRect.top)) / 3)
            + GetMenuBarHeight());
    return(true);
}

/*-----------------------------DIALOG FILTER

    DialogFilter() allows exit of ModalDialog() on any
mouseDown or keyDown event. */

pascal Boolean DialogFilter(whichDialog, whichEvent,
    whichItem)
DialogPtr whichDialog;
EventRecord *whichEvent;
short *whichItem;
{
    WindowPtr whichWindow;

    if((whichEvent->what == mouseDown) ||
(whichEvent->what == keyDown)) {

        if((FindWindow(whichEvent->where, &whichWindow)
            == inMenuBar)
                || ((whichEvent->what == keyDown) &&
            (whichEvent->modifiers & cmdKey))) {
            /* Re-post event and ignore any error. */
            (void) PostEvent(whichEvent->what,
            whichEvent->message);
        }
        *whichItem = 1;
        return(true);
    }
    else {
        return(false);
    }
}

/*-----------------TEAROFF  PALETTE

    version 1.0
    by Don Melton and Mike Ritter

    Copyright (C)1987, 1988 by Impulse Technologies,
        Inc., all rights reserved.

    Filename:       Error.c
    Font:               Courier, 9 point
    Tab setting:        2
    Compiler:       LightspeedC 2.15
    Project type:       APPL
    Creator:        TOPD
    Segment:            Main */

/*GLOBAL CONSTANT DEFINITIONS AND VARIABLE DECLARATIONS */

#include "Constants.h"
#include "Variables.h"

/*-----------------------EXTERNAL FUNCTION DECLARATIONS */

/* None */

/*-----------------------------FORWARD FUNCTION DECLARATIONS */

short GoodNewPtr();
short GoodNewHandle();
short GoodResource();

pascal long RecoverMemory();

/*-----------------------------GOOD NEW POINTER */

short GoodNewPtr(whichPtr)
Ptr whichPtr;
{
    if((whichPtr != nil) && (MemErr == noErr)) {
        return(true);
    }
    return(false);
}

/*-----------------------------GOOD NEW HANDLE */

short GoodNewHandle(whichHandle)
Handle whichHandle;
{
    if(whichHandle != nil) {
        return(GoodNewPtr(*whichHandle));
    }
    return(false);
}


/*-----------------------------GOOD RESOURCE */

short GoodResource(whichHandle)
Handle whichHandle;
{
    if((whichHandle != nil) && (ResError() == noErr)) {

        if(*whichHandle != nil) {
            return(true);
        }
    }
    return(false);
}


/*-----------------------------RECOVER MEMORY

    RecoverMemory() is a very simple grow zone function
designed to prevent the ROM or resident software such as
desk accessories from generating out of memory errors. After
this function is called, the application will quit before
processing the next event.

    See 'Inside Macintosh Volume II,' pages 42-43, about
setting up grow zone functions. */
```

```
pascal long RecoverMemory(memoryNeeded)
Size memoryNeeded;
{
    long bufferSize;

    SetUpA5();

    bufferSize = GetHandleSize(MemoryBuffer);

    if(bufferSize != 0) {
        DisposHandle(MemoryBuffer);
    }
    OutOfMemory = true;

    RestoreA5();

    return(bufferSize);
}

/*-------------------------TEAROFF  PALETTE

    version 1.0
    by Don Melton and Mike Ritter

    Copyright (C)1987, 1988 by Impulse Technologies,
        Inc., all rights reserved.

    Filename:       Initialize.c
    Font:               Courier, 9 point
    Tab setting:        2
    Compiler:       LightspeedC 2.15
    Project type:   APPL
    Creator:        TOPD
    Segment:            Initialize

    IMPORTANT! Use ResEdit to set the code resource of this
segment to unlocked. */

/*-GLOBAL CONSTANT DEFINITIONS AND VARIABLE DECLARATIONS */

#include "Constants.h"
#include "Variables.h"

/*-------------------------EXTERNAL FUNCTION DECLARATIONS */

extern short GoodNewPtr();          /* Error.c */
extern short GoodNewHandle();           /* Error.c */
extern short GoodResource();            /* Error.c */
extern pascal long RecoverMemory();     /* Error.c */

extern pascal void DrawToolMenu();      /* Menu.c */
extern pascal short FindToolItem();     /* Menu.c */
extern pascal void HiliteToolItem();/* Menu.c */
extern pascal void DrawPatternMenu();   /* Menu.c */
extern pascal short FindPatternItem();  /* Menu.c */
extern pascal void HilitePatternItem(); /* Menu.c */
extern pascal void DrawColorMenu();     /* Menu.c */
extern pascal short FindColorItem();/* Menu.c */
extern pascal void HiliteColorItem();   /* Menu.c */

/*-------------------------FORWARD FUNCTION DECLARATIONS */

void Initialize();

void SetupEnvironment();
void SetupMemory();

void SetupPalettes();
void SetupToolPalette();
void SetupPatternPalette();
void SetupColorPalette();
void SetupPaletteRects();

void SetupMenus();

/*-------------------------INITIALIZE */

void Initialize() {

    InitGraf(&thePort);
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs(nil);
    InitCursor();

    FlushEvents(everyEvent, 0);

    SetupEnvironment();
    SetupMemory();
    SetupPalettes();
    SetupMenus();

    TopWindow = TopPalette = TopDocument = nil;
    BottomPalette = BRING_TO_FRONT;

    PositionCounter = 0;

    Sleep = SLEEP_DURATION;

    ClosingAll = Quitting = Finished = false;
}

/*-------------------------SETUP ENVIRONMENT */

void SetupEnvironment() {

    if(SysEnvirons(VERSION_REQUESTED, &Environment)
            == envBadSel) {
        ExitToShell();
    }
    if(Environment.machineType >= envMachUnknown) {
        WNEIsImplemented =
                NGetTrapAddress
    (WAIT_NEXT_EVENT_TRAP_NUMBER,ToolTrap)
                != NGetTrapAddress
                (UNIMPLEMENTED_TRAP_NUMBER, ToolTrap);
    }
    else {
        WNEIsImplemented = false;
    }
}

/*-------------------------SETUP MEMORY */

void SetupMemory() {

    MaxApplZone();

    MoreMasters();
    MoreMasters();

    if(!GoodNewHandle(MemoryBuffer =
NewHandle(MEMORY_BUFFER_SIZE))) {
        ExitToShell();
    }
    OutOfMemory = false;

    /* Setup a simple grow zone function. */
    SetGrowZone(RecoverMemory);
}

/*-------------------------SETUP PALETTES */

void SetupPalettes() {
    short index;
    TearOffMGlobalsPtr whichTearOff;
```

```
    Rect windowRects[PALETTE_COUNT];
    short attributes;

    for(index = 0; index < PALETTE_COUNT; index++) {

        /* Get a handle to TearOffMenuGlobals. */
        if(!GoodResource(TearOffs[index]
                = (TearOffMGlobalsHdl)
                GetResource(TEAR_OFF_MENU_GLOBALS_TYPE,
                TOOL_MENU_ID + index))) {
            ExitToShell();
        }
        MoveHHi(TearOffs[index]);
        HLock(TearOffs[index]);
        whichTearOff = *TearOffs[index];

        switch(index) {

            case TOOL_PALETTE:
                SetupToolPalette(whichTearOff,
                &windowRects[index]);
                break;

            case PATTERN_PALETTE:
                SetupPatternPalette(whichTearOff,
                &windowRects[index]);
                break;

            case COLOR_PALETTE:
                SetupColorPalette(whichTearOff,
                &windowRects[index]);
        }
        if(Environment.hasColorQD) {
            /* Create a color port. */
            if(!GoodNewPtr(Palettes[index]
                    = NewCWindow(nil,
                    &windowRects[index],
                    NULL,
                    NOT_VISIBLE,
                    /* Set procedure ID based on
            PaletteWDEF resource ID. */
                    (16 * PALETTE_WDEF_ID)
                    + noGrowDocProc,
                    BRING_TO_FRONT,
                    GO_AWAY_BOX,
                    (long) index))) {
                ExitToShell();
            }
        }
        else {
            /* Create an old-style port. */
            if(!GoodNewPtr(Palettes[index]
                    = NewWindow(nil,
                    &windowRects[index],
                    NULL,
                    NOT_VISIBLE,
                    /* Set procedure ID based on
            PaletteWDEF resource ID. */
                    (16 * PALETTE_WDEF_ID)
                    + noGrowDocProc,
                    BRING_TO_FRONT,
                    GO_AWAY_BOX,
                    (long) index))) {
                ExitToShell();
            }
        }
        /* Finish Initializing TearOffMenuGlobals
structure. */
        whichTearOff->environment = &Environment;
        whichTearOff->paletteWindow = Palettes[index];

        HUnlock(TearOffs[index]);
    }
}
```

```
/*-------------------------------SETUP TOOL PALETTE */

void SetupToolPalette(whichTearOff, windowRect)
TearOffMGlobalsPtr whichTearOff;
Rect *windowRect;
{
    PicHandle toolPicture;

    if(!GoodResource(toolPicture =
GetPicture(TOOL_PICT_ID))) {
        ExitToShell();
    }
    HNoPurge(toolPicture);

    /* Calculate palette's portRect from toolPicture. */
    *windowRect = (*toolPicture)->picFrame;
    OffsetRect(windowRect, -windowRect->left,
    -windowRect->top);

    SetupPaletteRects(&ToolRects,
            TOOLS_ACROSS,
            TOOLS_DOWN,
            (windowRect->right + 1) / TOOLS_ACROSS,
            (windowRect->bottom + 1) / TOOLS_DOWN);

    /* Initialize the TearOffMenuGlobals structure. */
    whichTearOff->currentItem = DEFAULT_TOOL;
    whichTearOff->drawMenuProc = DrawToolMenu;
    whichTearOff->findItemProc = FindToolItem;
    whichTearOff->hiliteItemProc = HiliteToolItem;
}

/*-------------------------------SETUP PATTERN PALETTE */

void SetupPatternPalette(whichTearOff, windowRect)
TearOffMGlobalsPtr whichTearOff;
Rect *windowRect;
{
/* Calculate the portRect from number of patterns. */
    windowRect->left = 0;
    windowRect->top = 0;
    windowRect->right = (PATTERNS_ACROSS
            * (PATTERN_ITEM_WIDTH + 1)) + 1;
    windowRect->bottom = (PATTERNS_DOWN
            * (PATTERN_ITEM_HEIGHT + 1)) + 1;

    SetupPaletteRects(&PatternRects,
            PATTERNS_ACROSS,
            PATTERNS_DOWN,
            PATTERN_ITEM_WIDTH + 1,
            PATTERN_ITEM_HEIGHT + 1);

    /* Initialize the TearOffMenuGlobals structure. */
    whichTearOff->currentItem = DEFAULT_PATTERN;
    whichTearOff->drawMenuProc = DrawPatternMenu;
    whichTearOff->findItemProc = FindPatternItem;
    whichTearOff->hiliteItemProc = HilitePatternItem;
}

/*-------------------------------SETUP COLOR PALETTE */

void SetupColorPalette(whichTearOff, windowRect)
TearOffMGlobalsPtr whichTearOff;
Rect *windowRect;
{
/* Calculate the portRect from the number of colors. */
    windowRect->left = 0;
    windowRect->top = 0;
    windowRect->right = (COLORS_ACROSS
            * (COLOR_ITEM_WIDTH + 1)) + 1;
    windowRect->bottom = (COLORS_DOWN
            * (COLOR_ITEM_HEIGHT + 1)) + 1;
```

```
    SetupPaletteRects(&ColorRects,
            COLORS_ACROSS,
            COLORS_DOWN,
            COLOR_ITEM_WIDTH + 1,
            COLOR_ITEM_HEIGHT + 1);

    /* Initialize the TearOffMenuGlobals structure. */
    whichTearOff->currentItem = DEFAULT_COLOR;
    whichTearOff->drawMenuProc = DrawColorMenu;
    whichTearOff->findItemProc = FindColorItem;
    whichTearOff->hiliteItemProc = HiliteColorItem;
}

/*-----------------------------SETUP PALETTE RECTS */

void SetupPaletteRects(whichRects, itemsAcross,
itemsDown, itemWidth, itemHeight)
Rect *whichRects;
short itemsAcross;
short itemsDown;
short itemWidth;
short itemHeight;
{
    short index;
    short across;
    short down;

    whichRects->left = 0;
    whichRects->top = 0;
    whichRects->right = 0;
    whichRects->bottom = 0;

    index = 1;

    for(down = 0; down < itemsDown; down++) {

        for(across = 0;
                across < itemsAcross; across++) {
            (whichRects + index)->left
                    = across * itemWidth;
            (whichRects + index)->top
                    = down * itemHeight;
            (whichRects + index)->right
                    = (across + 1) * itemWidth;
            (whichRects + index)->bottom
                    = (down + 1) * itemHeight;
            index += 1;
        }
    }
}

/*-----------------------------SETUP MENUS */

void SetupMenus() {
    short index;

    for(index = APPLE_MENU_INDEX;
            index < MENU_COUNT - 1; index++) {
        Menus[index] = GetMenu(index + MENU_ID_OFFSET);
        InsertMenu(Menus[index], 0);
    }
    AddResMenu(Menus[APPLE_MENU_INDEX], 'DRVR');

    ColorMenuVisible = false;

    if(Environment.hasColorQD) {
        MaxDevice = GetMaxDevice(&(*GrayRgn)->rgnBBox);

        Menus[COLOR_MENU_INDEX] =
    GetMenu(COLOR_MENU_ID);

        if((*(*MaxDevice)->gdPMap)->pixelSize == 8) {
            /* Insert Color menu if 256 colors. */
            InsertMenu(Menus[COLOR_MENU_INDEX], 0);
            ColorMenuVisible = true;
        }
    }
    DrawMenuBar();
}
```

```
/*PALETTE WINDOW DEFINITION

      version 1.0
      by Don Melton and Mike Ritter

      Copyright (C)1987, 1988 by Impulse Technologies, Inc., all
rights reserved.

      Filename:    PaletteWDEF.c
      Font:        Monaco, 9 point
      Tab setting: 2
      Compiler:    LightspeedC 2.15,
      Project type: WDEF, ID: 128, Name: (None)

      IMPORTANT! Use ResEdit to set the WDEF resource to non-
purgeable.

-------------------------

DESCRIPTION

      PaletteWDEF is a WDEF code resource designed to display
windows that mimic the appearance of tear-off palettes in
HyperCard. However, PaletteWDEF will unhilite a title bar, and
PaletteWDEF windows have a 3 pixel indented drop shadow just like
a menu.

      To use PaletteWDEF, an application must calculate the
procedure ID of the window to be created, at runtime or store it
in a WIND or DLOG resource. With PaletteWDEF's resource ID being
128, the folllowing code will create a new palette window:

      w = NewWindow(p, bounds, title, visible, (16 * 128) +
noGrowDocProc, behind, goAwayFlag, refCon);

      See 'Inside Macintosh Volume I,' page 298, for more
information on calculating the procedure ID.

      Note: Although NewCWindow() or GetNewCWindow() can be used
with PaletteWDEF, it does not support the use of wctb resources
for color information, or respond visibly to SetWinColor().

      Note: PaletteWDEF will not draw a grow region or zoom box
in a window. It completely ignores any and all variation codes
passed to it. */

/*-----------------------------INCLUDE DEFINITIONS */

#include "MacTypes.h"
#include "MemoryMgr.h"
#include "Quickdraw.h"
#include "WindowMgr.h"

/*-----------------------------GLOBAL CONSTANT DEFINITIONS */

typedef struct QuickDraw {
      char private[76];
      long randSeed;
      BitMap screenBits;
      Cursor arrow;
      Pattern dkGray;
      Pattern ltGray;
      Pattern gray;
      Pattern black;
      Pattern white;
      GrafPtr thePort;
} QuickDraw;

#define PALETTE_TITLE_BAR_HEIGHT 10
#define PALETTE_SHADOW_INDENT 3
```

```
/*-----------------------------GLOBAL VARIABLE DECLARATIONS */

/* None, even though the LightspeedC compiler allows it, others
don't. */

/*-------------------------EXTERNAL FUNCTION DECLARATIONS */

/* None, they're not allowed. */

/*-------------------------FORWARD FUNCTION DECLARATIONS */

pascal long main();
void DrawWindow();
long TestWindowHit();
void CalcWindowRegions();

/*-----------------------------PALETTE WINDOW DEFINITION

      The main function is called by the Window Manager in the
Macintosh ROM. It's designed to repsond to three messages: drawing
the window, testing which part of the window has been hit, and
calculating the various window regions.

      See 'Inside Macintosh Volume I,' pages 297-302, for more
information on this function. */

pascal long main(variation, whichWindow, message, parameter)
short variation;
WindowPtr whichWindow;
short message;
long parameter;
{
      long result;

      result = 0;

      /* Use multiple 'if's rather than 'switch' to avoid linking
of 400+ bytes of runtime code. */

      if(message == wDraw) {
        DrawWindow(whichWindow, parameter);
      }
      else if(message == wHit) {
        result = TestWindowHit(whichWindow, parameter);
      }
      else if(message == wCalcRgns) {
        CalcWindowRegions(whichWindow);
      }
      return(result);
}

/*-----------------------------DRAW WINDOW */

void DrawWindow(whichWindow, parameter)
WindowPtr whichWindow;
long parameter;
{
      Rect frame;
      Rect goAway;
      PenState savePen;
      short tone;
      short index;
      Pattern hilite;
      QuickDraw *qdGlobals;

      /* Set up a pointer to QuickDraw's global variables. */
      qdGlobals = (QuickDraw *)(*(Byte **)CurrentA5
          - (sizeof(QuickDraw) - sizeof(GrafPtr)));

      if(((WindowPeek) whichWindow)->visible) {
        /* Calculate the window frame. */
        frame = (*((WindowPeek) whichWindow)->strucRgn)-
>rgnBBox;
        frame.bottom -= 1;
        frame.right -= 1;

        /* Calculate the goAway box. */
        goAway = frame;
        goAway.top += 2;
        goAway.left += 8;
        goAway.bottom = goAway.top + 7;
        goAway.right = goAway.left + 7;

        if(parameter != 0) {
          /* Hilite the goAway box. */
          InvertRect(&goAway);
        }
        else {
          /* Draw the window frame, drop shadow, title bar and
goAway box. */
          GetPenState(&savePen);
          PenNormal();

          /* Draw the frame. */
          FrameRect(&frame);

          /* Draw the drop shadow. */
          MoveTo(frame.left + PALETTE_SHADOW_INDENT,
frame.bottom);
          LineTo(frame.right, frame.bottom);
          LineTo(frame.right, frame.top +
PALETTE_SHADOW_INDENT);

          /* Calculate and draw the title bar. */
        frame.bottom = frame.top + (PALETTE_TITLE_BAR_HEIGHT + 1);
          FrameRect(&frame);
          InsetRect(&frame, 1, 1);

          if(((WindowPeek) whichWindow)->hilited) {
            /* Adjust pattern and fill title bar. */
            tone = frame.left & 1 ? 0x55 : 0xaa;

            for(index = 0; index <= 7; index++) {
          hilite[index] = (index + frame.top) & 1 ? tone : 0;
            }
            FillRect(&frame, &hilite);

            /* Draw the goAway box. */
            frame = goAway;
            InsetRect(&frame, -1, -1);
            EraseRect(&frame);
            FrameRect(&goAway);
          }
          else {
            FillRect(&frame, qdGlobals->white);
          }
          SetPenState(&savePen);
        }
      }
}

/*-----------------------------TEST WINDOW HIT */

long TestWindowHit(whichWindow, parameter)
WindowPtr whichWindow;
long parameter;
{
      Point where;
      Rect drag;
      Rect goAway;

      where.v = parameter >> 16; /* HiWord(parameter). */
      where.h = parameter & 0xFFFF; /* LoWord(parameter). */

      if(PtInRect(where, &(*((WindowPeek) whichWindow)-
>contRgn)->rgnBBox)) {
        return(wInContent);
```


© The Definitive MacTutor, Vol. 4

```
      }
      else {
        /* The mouse is in the window frame. */
        drag   =   (*((WindowPeek)   whichWindow)->strucRgn)-
>rgnBBox;
        drag.bottom = drag.top + (PALETTE_TITLE_BAR_HEIGHT + 1);
        drag.right -= 1;

        if(PtInRect(where, &drag)) {
          goAway = drag;
          goAway.top += 2;
          goAway.left += 8;
          goAway.bottom = goAway.top + 7;
          goAway.right = goAway.left + 7;

          if((((WindowPeek)   whichWindow)->hilited)   &&
(PtInRect(where, &goAway))) {
            return(wInGoAway);
          }
          return(wInDrag);
        }
      }
      return(wNoHit);
}

/*
-------------------------
CALCULATE WINDOW REGIONS */

void CalcWindowRegions(whichWindow)
WindowPtr whichWindow;
{
      Rect windowRect;
      RgnHandle shadowRgn;

      /* Calculate the content region of the window. */
      windowRect = whichWindow->portRect;
      OffsetRect(&windowRect,     -     whichWindow-
>portBits.bounds.left, - whichWindow->portBits.bounds.top);
      RectRgn(((WindowPeek)  whichWindow)->contRgn,  &window-
Rect);

      /* Calculate the structure region of the window. */
      /* First, calculate the window frame. */
      windowRect.top -= PALETTE_TITLE_BAR_HEIGHT + 1;
      windowRect.left -= 1;
      windowRect.bottom += 1;
      windowRect.right += 1;
      RectRgn(((WindowPeek)  whichWindow)->strucRgn,  &window-
Rect);

      /* Next, calculate the frame's drop shadow. */
      windowRect.top += PALETTE_SHADOW_INDENT;
      windowRect.left += PALETTE_SHADOW_INDENT;
      windowRect.bottom += 1;
      windowRect.right += 1;
      RectRgn(shadowRgn = NewRgn(), &windowRect);

      /* Combine the frame and the shadow to calculate the
structure. */
      UnionRgn(((WindowPeek) whichWindow)->strucRgn, shadowRgn,
((WindowPeek) whichWindow)->strucRgn);

      DisposeRgn(shadowRgn);
}
```

```
/*----------------TEAROFF  MENU  DEFINITION

   version 1.0
   by Don Melton and Mike Ritter

   Copyright (C)1987, 1988 by Impulse Technologies,
   Inc., all rights reserved.

   Filename:        TearOffMDEF.c
   Font:            Monaco, 9 point
   Tab setting:     2
   Compiler:        LightspeedC 2.15
   Project type: MDEF
   ID:              128
   Name:            (None)

   IMPORTANT! Use ResEdit to set the MDEF resource to non-
purgeable.

   IMPORTANT! Use ResEdit or FEdit to change hex string
'21c809ce' at hex offset 14 in the MDEF resource to
'4e714e71.' This substitutes two 'nop' instructions for a
'move.l a0,ToolScratch.'

-------------------------
DESCRIPTION

   TearOffMDEF is an MDEF code resource implementing the
common actions of tear-off menus for any application. It's
designed to communicate with an application via a TearOff-
MenuGlobals structure in a TOMG resource. At runtime, Tea-
rOffMDEF uses the menuID of the MenuInfo, passed to it by the
Macintosh ROM, to find a TOMG resource of the same ID. The
TearOffMenuGlobals structure contains eight elements:

      drawMenuProc     A function pointer to a menu in global
coordinates within the WMgrPort.

      findItemProc     A function pointer to a pascal-type
procedure which returns the number of the menu item where the
mouse is currently located.

      hiliteItemProc   A function pointer to a pascal-type
procedure which hilites or unhilites a given menu item.

      environment      A pointer to a SysEnvRec used to test
if the 64K ROMs are present.

      paletteWindow    A WindowPtr used to calculate
menuWidth, menuHeight, and the structure boundary from the
window's portRect.

      currentItem      A short int containing the number of
the currently hilited menu item.

      position         A Point passed to the application from
TearOffMDEF indicating the top, left point, in global coordi-
nates, of a torn-off menu.

      itemHilited      A Boolean used internally by Tea-
rOffMDEF.

   All of these elements, except position and itemHilited,
must be initialized by the application before a tear-off menu
is inserted into the menu list.

   IMPORTANT! The menuProc field of a tear-off MENU resource
must equal TearOffMDEF's resource ID.

   The application is completely responsible for the appear-
ance of the menu. TearOffMDEF does not contain code to draw
the contents of a menu, nor does it contain code to hilite
menu items. A set of the three following procedures must be
declared in the application for each tear-off menu:

      pascal void DrawMenuProc(destRect)
      Rect *destRect;   /* The menu rectangle in the
                  current grafPort. */

      pascal short FindItemProc(mousePt)
      Point mousePt;   /*The point in which the mouse
                  is currently located, relative
```

```
                to the top,left of the menu
             rectangle.

                    The number of the menu item
                 where the mouse is currently
              located should bereturned to
              TearOffMDEF by the application.
              */

      pascal void HiliteItemProc(destRect, item, hilite)
      Rect *destRect;  /* The menu rectangle in the
                  current grafPort. */
      short item;      /* The number of the menu item
                  to be hilited or unhilited.
      Boolean hilite;  /* A flag indicating the hilite
                     state. Set true to hilite and
                  false to unhilite.

   IMPORTANT! The segment in which these procedures are
declared must remain locked at runtime.

   It's very simple for the application to call these same
procedures to draw into the window once the user has torn it
off.

   TearOffMDEF will return -1 to the Menu Manager if a menu
has been torn off. It's the application's responsibility to
move the window into position and make it visible.

   Note: TearOffMDEF is not designed to implement pop-up
menus. */

/*-------------------------------INCLUDE DEFINITIONS */

#include "MacTypes.h"
#include "MemoryMgr.h"
#include "MenuMgr.h"
#include "OSUtil.h"
#include "pascal.h"
#include "Quickdraw.h"
#include "ResourceMgr.h"
#include "WindowMgr.h"

/*-------------------------------GLOBAL CONSTANT DEFINITIONS */

#define nil 0

typedef struct QuickDrawGlobals {
   char private[76];
   long randSeed;
   BitMap screenBits;
   Cursor arrow;
   Pattern dkGray;
   Pattern ltGray;
   Pattern gray;
   Pattern black;
   Pattern white;
   GrafPtr thePort;
} QuickDrawGlobals;

typedef struct TearOffMenuGlobals {
   void (*drawMenuProc)();
   short (*findItemProc)();
   void (*hiliteItemProc)();
   SysEnvRec *environment;
   WindowPtr paletteWindow;
   Point position;
   short currentItem;
   Boolean itemHilited;
} TearOffMenuGlobals, *TearOffMGlobalsPtr, **TearOffMGlobal-
sHdl;

#define TEAR_OFF_MENU_GLOBALS_TYPE 'TOMG'

#define TEAR_OFF_MARGIN 15
#define MOVE_PALETTE_ITEM (-1)

#define PALETTE_TITLE_BAR_HEIGHT 10
#define PALETTE_SHADOW_INDENT 3
#define PALETTE_OFFSET 5

/*-------------------------------GLOBAL VARIABLE DECLARATIONS */

/* None, even though the LightspeedC compiler allows it,
others don't. */

/*-------------------------------EXTERNAL FUNCTION DECLARATIONS */

/* None, they're not allowed. */

/*-------------------------------FORWARD FUNCTION DECLARATIONS */

pascal void main();
void ChooseItem();
void DragMenu();

/*-------------------------------TEAROFF MENU DEFINITION

   The main function is called by the Menu Manager in the
Macintosh ROM. It's designed to repsond to three messages:
drawing the menu, choosing an item, and  calculating the menu
size. It will not respond to a pop-up menu request. Before it
dispatches to these routines, it first checks to see if the
TearOffMenuGlobals structure is available on the heap,  and
locks it down. It does nothing if it cannot find this struc-
ture.

   See 'Inside Macintosh Volume I,' pages 362-363, for more
information on this function. */

pascal void main(message, whichMenu, menuRect, hitPt,
whichItem)
short message;
MenuHandle whichMenu;
Rect *menuRect;
Point hitPt;
short *whichItem;
{
   TearOffMGlobalsHdl whichTOMGlobalsHdl;
   TearOffMGlobalsPtr tearOffMGlobals;

   /* Make sure A5 is valid so the application code can
use it's own global variables. */
   SetUpA5();

   /* Use the menuID to get a handle to the correct  TearOff-
MenuGlobals structure. */
   whichTOMGlobalsHdl = (TearOffMGlobalsHdl)
GetResource(TEAR_OFF_MENU_GLOBALS_TYPE,
         (*whichMenu)->menuID);

   if((ResErr == noErr) && (whichTOMGlobalsHdl != nil)) {
      /* Lock the TearOffMenuGlobals structure so it can
   be dereferenced safely. */
      HLock(whichTOMGlobalsHdl);
      tearOffMGlobals = *whichTOMGlobalsHdl;

      /* Dispatch according to the message received from
   the Macintosh ROM.  Use multiple 'if's
      rather than 'switch' to avoid linking of 400+
bytes of runtime code. */

      if(message == mDrawMsg) {
         CallPascal(menuRect,              (tearOffMGlobals)-
>drawMenuProc);
         /* Hilite currentItem here instead of in
   ChooseItem(). It may not be called son. */
         CallPascal(menuRect,              (tearOffMGlobals)-
```

```
>currentItem, true,
                (tearOffMGlobals)->hiliteItemProc);
            /* Let ChooseItem() know currentItem has been
        hilited. */
            (tearOffMGlobals)->itemHilited = true;
        }
        else if(message == mChooseMsg) {
            ChooseItem(tearOffMGlobals, menuRect, hitPt,
                whichItem);
        }
        else if(message == mSizeMsg) {
            /* Calculate the menuRect from the
paletteWindow's portRect. */
            (*whichMenu)->menuWidth
                  = ((tearOffMGlobals)->paletteWindow)
                  ->portRect.right
                  - ((tearOffMGlobals)->paletteWindow)
                  ->portRect.left;
                  (*whichMenu)->menuHeight
                  = ((tearOffMGlobals)->paletteWindow)
                  ->portRect.bottom
                  - ((tearOffMGlobals)->paletteWindow)
                  ->portRect.top;
        }
        HUnlock(whichTOMGlobalsHdl);
    }
    RestoreA5();
}

/*-------------------------CHOOSE ITEM

    If the mouse is within the menuRect, ChooseItem() will call
the application to find in which item the mouse is currently
located. The application is also called to hilite or unhilite
a single item. If the mouse is outside the menuRgn ChooseItem
will call DragMenu(). The menuRgn is defined as an area made
up of the menuRect with a margin and the menu bar. */

void ChooseItem(tearOffMGlobals, menuRect, hitPt,
whichItem)
TearOffMGlobalsPtr tearOffMGlobals;
Rect *menuRect;
Point hitPt;
short *whichItem;
{
    short saveItem;
    short hiliteItem;
    Point mousePt;
    Rect marginRect;
    RgnHandle menuRgn;
    RgnHandle tempRgn;
    QuickDrawGlobals *qdGlobals;

    /* Set up pointer to QuickDraw's global variables. */
    qdGlobals = (QuickDrawGlobals *)(*(Byte **)CurrentA5
          - (sizeof(QuickDrawGlobals)
          - sizeof(GrafPtr)));

    saveItem = *whichItem;
    hiliteItem = *whichItem = 0;

    /* If the hitPt is empty, the Menu Manager is calling
TearOffMDEF repeatedly to flash the item. */

    if((hitPt.h != 0) && (hitPt.v != 0)) {

        if(PtInRect(hitPt, menuRect)) {
            mousePt.h = hitPt.h - menuRect->left;
            mousePt.v = hitPt.v - menuRect->top;

            hiliteItem = *whichItem = CallPascalW(mousePt,
                (tearOffMGlobals)->findItemProc);

            if((tearOffMGlobals)->itemHilited) {
                saveItem = (tearOffMGlobals)->currentItem;
                (tearOffMGlobals)->itemHilited = false;
            }
        }
        else {

            if(!(tearOffMGlobals)->itemHilited) {
                hiliteItem = (tearOffMGlobals)->currentItem;
                (tearOffMGlobals)->itemHilited = true;
            }
        }
    }
    if(saveItem != hiliteItem) {
        /* Unhilite the old item. */
        CallPascal(menuRect, saveItem, false,
(tearOffMGlobals)->hiliteItemProc);
        /* Hilite the new item. */
        CallPascal(menuRect, hiliteItem, true,
    (tearOffMGlobals)->hiliteItemProc);
    }
    /* Calculate the menu region. if the mouse is outside
this region tear off the menu. */
    marginRect = *menuRect;
    marginRect.left -= TEAR_OFF_MARGIN;
    marginRect.bottom += TEAR_OFF_MARGIN;
    marginRect.right += TEAR_OFF_MARGIN;
    RectRgn(menuRgn = NewRgn(), &marginRect);

    RectRgn(tempRgn = NewRgn(), &qdGlobals->screenBits.bounds);

    if(tearOffMGlobals->environment->machineType
          < envMachUnknown) {
        /* Arrggh! Someone has the 64K ROMs. */
        (*tempRgn)->rgnBBox.bottom = 20;
    }
    else {
        (*tempRgn)->rgnBBox.bottom = MBarHeight;
    }
    UnionRgn(menuRgn, tempRgn, menuRgn);

    if(!PtInRgn(hitPt, menuRgn)) {
        DragMenu(tearOffMGlobals, hitPt, whichItem,
        menuRgn, tempRgn, qdGlobals);
    }
    DisposeRgn(tempRgn);
    DisposeRgn(menuRgn);
}

/*-------------------------DRAG MENU

    DragMenu() draws an outline of the paletteWindow as the
mouse is moved outside the menuRgn. If the mouse button is
released outside this region, DragMenu() will set whichItem to
-1 and exit. */

void DragMenu(tearOffMGlobals, hitPt, whichItem,
menuRgn, saveRgn, qdGlobals)
TearOffMGlobalsPtr tearOffMGlobals;
Point hitPt;
short *whichItem;
RgnHandle menuRgn;
RgnHandle saveRgn;
QuickDrawGlobals *qdGlobals;
{
    PenState savePen;
    long finalTicks;
    Rect windowRect;
    RgnHandle structureRgn;
    short hOffset;
    Point oldMouse;
    Point newMouse;
    RgnHandle dragRgn;

    GetPenState(&savePen);
```

```
   PenSize(1, 1);
   PenMode(notPatXor);
   PenPat(qdGlobals->gray);

   /* Since paletteWindow may not be visible, its     portRect
must be used to calculate structureRgn. First, calculate the
window frame. */
   windowRect
         = (tearOffMGlobals->paletteWindow)->portRect;

   /* Correct windowRect if top, left of portRect     isn't
0,0. */
   OffsetRect(&windowRect, -windowRect.left,
- windowRect.top);

   windowRect.right += 3;
   windowRect.bottom += PALETTE_TITLE_BAR_HEIGHT + 3;
   RectRgn(structureRgn = NewRgn(), &windowRect);

   hOffset = windowRect.right / 2;

   /* Next, calculate the frame's drop shadow. */
   windowRect.top += PALETTE_SHADOW_INDENT;
   windowRect.left += PALETTE_SHADOW_INDENT;
   windowRect.bottom += 1;
   windowRect.right += 1;
   RectRgn(saveRgn, &windowRect);

   /* Combine frame, shadow to calculate structure. */
   UnionRgn(structureRgn, saveRgn, structureRgn);

   GetClip(saveRgn);
   SetClip(GrayRgn);

   oldMouse = newMouse = hitPt;

   CopyRgn(structureRgn, dragRgn = NewRgn());
   OffsetRgn(dragRgn, newMouse.h - hOffset, newMouse.v
         - PALETTE_OFFSET);

   /* Draw first drag outline. */
   FrameRgn(dragRgn);

   /* Mimic DragGrayRgn() by staying in a loop until the
mouse button is released or the mouse is inside the   menuRgn
again. */

   do {
      /* Don't erase old drag outline if the mouse
hasn't moved. */

      if(!EqualPt(newMouse, oldMouse)) {
         /* Erase old drag outline. */
         FrameRgn(dragRgn);

         CopyRgn(structureRgn, dragRgn);
         OffsetRgn(dragRgn, newMouse.h - hOffset,
         newMouse.v - PALETTE_OFFSET);

         /* Draw new drag outline. */
         FrameRgn(dragRgn);
      }
      /* Keep drag outline from flickering. */
      Delay(2, &finalTicks);

      oldMouse = newMouse;
      GetMouse(&newMouse);
   }
   while(WaitMouseUp() && !PtInRgn(newMouse, menuRgn));

   /* Erase final drag outline. */
   FrameRgn(dragRgn);

   if(!PtInRgn(newMouse, menuRgn)) {
      (tearOffMGlobals)->position.h =                (*dragRgn)-
>rgnBBox.left + 1;
      (tearOffMGlobals)->position.v =                (*dragRgn)-
>rgnBBox.top + 11;

      /* Tell application the menu was torn off. */
      *whichItem = MOVE_PALETTE_ITEM;
   }
   DisposeRgn(structureRgn);
   DisposeRgn(dragRgn);

   SetClip(saveRgn);
   SetPenState(&savePen);
}
```

```
/*-------------------------
T E A R O F F   P A L E T T E

  version 1.0
  by Don Melton and Mike Ritter

  Copyright (C)1987, 1988 by Impulse Technologies,
  Inc., all rights reserved.

  Filename:       TearOffPalette.project.r
  Font:           Monaco, 9 point
  Tab setting:    2
  Compiler:       MPW Rez 2.0 */


/*-------------------------------INCLUDE TYPES */

#include "Types.r"

/*
-------------------------
TYPE DEFINITIONS */

type 'TOPD' as 'STR ';

/*-------------------------------RESOURCE DEFINITIONS */

resource 'BNDL' (128) {
  'TOPD',
  0,
  {
     'ICN#',
     {
        0, 128
     },
     'FREF',
     {
        0, 128
     }
  }
};

resource 'FREF' (128) {
  'APPL',
  0,
  ""
};

resource 'ICN#' (128, purgeable) {
   {
      /* Data */
      $"FFFF FFE0 8000 0030 8000 0030 EFBF FFF0"
      $"8880 0030 E8BF FFF0 8880 0030 EFBF FFF0"
      $"8000 0030 81FF FFFE FF00 0002 81BA AAAA"
      $"8128 0003 81BA AAAB 8100 0003 81FF FFFF"
      $"8100 0003 8100 0003 8100 0003 8100 0003"
      $"8100 0003 8100 0003 8100 0003 8100 0003"
      $"8100 0003 8100 0003 81FF FFFF 803F FFFF"
```

```
      $"8000 0030 8000 0030 FFFF FFF0 7FFF FFF0",
      /* Mask */
      $"FFFF FFE0 FFFF FFF0 FFFF FFF0 FFFF FFF0"
      $"FFFF FFF0 FFFF FFF0 FFFF FFF0 FFFF FFF0"
      $"FFFF FFF0 FFFF FFFE FFFF FFFE FFFF FFFE"
      $"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
      $"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
      $"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
      $"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
      $"FFFF FFF0 FFFF FFF0 FFFF FFF0 7FFF FFF0"
   }
};

resource 'PAT#' (128, preload) {
   {  /* 64 elements */
      $"",
      $"80",
      $"8000 0000 08",
      $"8800 0000 08",
      $"8800 0000 88",
      $"8800 2000 88",
      $"8800 2000 8800 02",
      $"8800 2200 8800 02",
      $"8800 2200 8800 22",
      $"A800 2200 8800 22",
      $"A800 2200 8A00 22",
      $"AA00 2200 8A00 22",
      $"AA00 2200 AA00 22",
      $"AA00 A200 AA00 22",
      $"AA00 A200 AA00 2A",
      $"AA00 AA00 AA00 AA",
      $"AA40 AA00 AA00 AA",
      $"AA40 AA00 AA04 AA",
      $"AA44 AA00 AA04 AA",
      $"AA44 AA00 AA44 AA",
      $"AA44 AA10 AA44 AA",
      $"AA44 AA10 AA44 AA01",
      $"AA44 AA11 AA44 AA01",
      $"AA44 AA11 AA44 AA11",
      $"AA54 AA11 AA44 AA11",
      $"AA54 AA11 AA45 AA11",
      $"AA55 AA11 AA45 AA11",
      $"AA55 AA11 AA55 AA11",
      $"AA55 AA51 AA55 AA11",
      $"AA55 AA51 AA55 AA15",
      $"AA55 AA55 AA55 AA15",
      $"AA55 AA55 AA55 AA55",
      $"EA55 AA55 AA55 AA55",
      $"EA55 AA55 AE55 AA55",
      $"EE55 AA55 AE55 AA55",
      $"EE55 AA55 EE55 AA55",
      $"EE55 BA55 EE55 AA55",
      $"EE55 BA55 EE55 AB55",
      $"EE55 BB55 EE55 AB55",
      $"EE55 BB55 EE55 BB55",
      $"FE55 BB55 EE55 BB55",
      $"FE55 BB55 EF55 BB55",
      $"FF55 BB55 EF55 BB55",
      $"FF55 BB55 FF55 BB55",
      $"FF55 FB55 FF55 BB55",
      $"FF55 FB55 FF55 BF55",
      $"FF55 FF55 FF55 BF55",
      $"FF55 FF55 FF55 FF55",
      $"FFD5 FF55 FF55 FF55",
      $"FFD5 FF55 FF5D FF55",
      $"FFDD FF55 FF5D FF55",
      $"FFDD FF55 FFDD FF55",
      $"FFDD FF75 FFDD FF55",
      $"FFDD FF75 FFDD FF57",
      $"FFDD FF77 FFDD FF57",
      $"FFDD FF77 FFDD FF77",
      $"FFFD FF77 FFDD FF77",
      $"FFFD FF77 FFDF FF77",
      $"FFFF FF77 FFDF FF77",
      $"FFFF FF77 FFFF FF77",
      $"FFFF FFF7 FFFF FF77",
      $"FFFF FFF7 FFFF FF7F",
      $"FFFF FFFF FFFF FF7F",
      $"FFFF FFFF FFFF FFFF"
   }
};

resource 'STR ' (128, preload) {
   "Untitled"
};

resource 'PICT' (128, preload) {
   1270,
   (0, 0, 83, 99),
   $"1101 A000 8201 000A 0000 0000 02D0 0240"
   $"9800 0E00 0000 0000 5300 6800 0000 0000"
   $"5300 6300 0000 0000 5300 6300 000C FE00"
   $"0680 0000 4000 0020 FD00 0CFE 0006 8000"
   $"0040 0000 20FD 000E FE00 0880 0000 4006"
   $"0020 0380 FF00 0F0B 0F3E 7080 1FC0 4069"
   $"C020 0280 FF00 0F0B 0800 1080 E030 4099"
   $"2020 0380 FF00 0F0B 0800 1081 0008 4099"
   $"2820 0380 FF00 0EFE 0008 8200 0840 4934"
   $"2003 80FF 000E FE00 0884 0008 4049 2420"
   $"0380 FF00 0F0B 0800 1084 0030 41A0 2420"
   $"0380 FF00 0F0B 0800 1084 01C0 4260 0420"
   $"1FF0 FF00 0F0B 0800 1083 8E00 4220 0820"
   $"1010 FF00 0F0B 0800 1086 7000 4100 0820"
   $"1FF0 FF00 0EFE 0008 8540 0040 8008 2010"
   $"10FF 000E FE00 0883 8000 4080 1020 1010"
   $"FF00 0F0B 0800 1080 8000 4040 1020 1010"
   $"FF00 0F06 0800 1080 8000 40FE 2001 1550"
   $"FF00 0F0B 0E7C F081 0000 4010 2020 2AB0"
   $"FF00 0EFE 0008 8000 0040 1020 207F E0FF"
   $"000C FE00 0680 0000 4000 0020 FD00 0CFE"
   $"0006 8000 0040 0000 20FD 0005 F5FF 01E0"
   $"000C FE00 0680 0000 4000 0020 FD00 0CFE"
   $"0006 8000 0040 0000 20FD 000D 0900 0F00"
   $"8000 0040 1C00 21FD 000E 0A00 0880 8000"
   $"0040 2200 2040 FE00 0E0A 0010 8080 07F0"
   $"4026 0021 15FE 000F 0B00 1900 8008 1040"
   $"2B00 2043 80FF 000F 0B00 2700 8010 3840"
   $"32C0 2104 40FF 000F 0B00 2200 8020 5840"
   $"2270 200F E0FF 000F 0B00 4200 8040 B040"
   $"4238 2008 20FF 000F 0B00 4400 8081 6040"
   $"851C 2009 E0FF 000F 0B00 8400 8102 C041"
   $"021C 2009 20FF 000F 0B00 8800 8205 8042"
   $"003C 2009 E0FF 000F 0B01 0800 87FB 0042"
   $"005C 2009 20FF 000F 0B01 1000 840E 0041"
   $"009C 2009 E0FF 000F 0B01 E000 840C 0040"
   $"811C 2009 E0FF 000F 0B01 C000 87F8 0040"
   $"421C 2008 20FF 000F 0B01 8000 8000 0040"
   $"2418 2008 20FF 000F 0B01 0000 8000 0040"
   $"1810 200F E0FF 000C FE00 0680 0000 4000"
   $"0020 FD00 0CFE 0006 8000 0040 0000 20FD"
   $"0005 F5FF 01E0 000C FE00 0680 0000 4000"
   $"0020 FD00 0CFE 0006 8000 0040 0000 20FD"
   $"000C FE00 0680 0000 4000 0020 FD00 0DFE"
   $"0007 8000 0040 0000 2003 FE00 0EFE 0008"
   $"87FF F840 1F80 200C C0FF 000E FE00 0884"
   $"0008 4060 6020 3030 FF00 0F0B 0C00 0084"
   $"0008 4180 1820 C00C FF00 0F0B 0300 0084"
   $"0008 4100 0820 8004 FF00 0F0B 00C0 0084"
   $"0008 4200 0420 8004 FF00 0F0B 0030 0084"
   $"0008 4200 0420 8004 FF00 0F0B 000C 0084"
   $"0008 4200 0420 8004 FF00 0F0B 0003 0084"
   $"0008 4200 0420 8004 FF00 0F0B 0000 C084"
   $"0008 4100 0820 8004 FF00 0F0B 0000 3084"
   $"0008 4180 1820 C00C FF00 0EFE 0008 8400"
   $"0840 6060 2030 30FF 000E FE00 0887 FFF8"
   $"401F 8020 0CC0 FF00 0DFE 0007 8000 0040"
   $"0000 2003 FE00 0CFE 0006 8000 0040 0000"
   $"20FD 000C FE00 0680 0000 4000 0020 FD00"
```

```
  $"0CFE 0006 8000 0040 0000 20FD 0005 F5FF"
  $"01E0 000C FE00 0680 0000 4000 0020 FD00"
  $"0CFE 0006 8000 0040 0000 20FD 000C FE00"
  $"0680 0000 4000 0020 FD00 0D09 07FF E080"
  $"0000 4000 0020 FD00 0D09 0618 6081 FFE0"
  $"4000 0020 FD00 0F0B 0418 2082 0010 40E0"
  $"7020 0FE0 FF00 0F0B 0418 2084 0008 4118"
  $"8820 1020 FF00 0E08 0018 0084 0008 4207"
  $"04FE 20FF 000F 0B00 1800 8400 0842 0004"
  $"2040 20FF 000F 0B00 1800 8400 0842 0004"
  $"2080 20FF 000F 0B00 1800 8400 0842 0004"
  $"2100 10FF 000F 0B00 1800 8400 0841 0008"
  $"2080 08FF 000F 0B00 1800 8400 0841 0010"
  $"2040 04FF 000F 0700 1800 8400 0840 80FE"
  $"2000 02FF 000F 0B00 1800 8200 1040 60C0"
  $"201F FEFF 000D 0900 1800 81FF E040 1F00"
  $"20FD 000D 0900 7E00 8000 0040 0000 20FD"
  $"000C FE00 0680 0000 4000 0020 FD00 0CFE"
  $"0006 8000 0040 0000 20FD 000C FE00 0680"
  $"0000 4000 0020 FD00 A000 83FF"
};

resource 'PICT' (129, purgeable) {
  858,
  (0, 0, 51, 166),
  $"1101 A000 8201 000A 0000 0000 02D0 0240"
  $"9800 1600 0000 0000 3300 A800 0000 0000"
  $"3300 A600 0000 0000 3300 A600 0006 FE00"
  $"0080 EF00 0703 0000 0140 EF00 0703 0000"
  $"0220 EF00 0703 0000 0410 EF00 0703 0000"
  $"0808 EF00 0703 0000 1004 EF00 0703 0000"
  $"2002 EF00 0703 0000 4001 EF00 0804 0000"
  $"8000 80F0 0008 0400 0100 0040 F000 0D04"
  $"0002 0000 20FB 0001 0180 F800 0D04 0004"
  $"0000 10FB 0001 0F80 F800 0D04 0008 001C"
  $"08FB 0001 3F80 F800 0C04 0010 00FC 04FB"
  $"0000 07F7 000C 0400 2000 FC02 FB00 0007"
  $"F700 0C04 0040 00FC 01FB 0000 07F7 000D"
  $"0500 8000 F800 80FC 0000 0FF7 000D 0501"
  $"0000 F800 40FC 0000 0EF7 000D 0502 0000"
  $"F800 20FC 0000 0EF7 000B 0304 0001 F0FA"
  $"0000 0EF7 0012 0E08 0001 F070 703C 787C"
  $"1C07 0E07 F01F FA00 130F 1000 01F1 F1F4"
  $"FCF8 FE7C 0F1E 1FF0 7F80 FB00 130F 2000"
  $"01E7 F3F3 FFF9 9EFC 0E1C 3871 C380 FB00"
  $"130F 4000 01EC E673 3E33 0F9C 0E1C 3031"
  $"8380 FB00 130F 8000 01E0 EC76 B83E 0E1C"
  $"0E1C 7023 8380 FB00 130F 4000 03C0 F87D"
  $"383C 0E3C 1E1C 7003 8780 FB00 120E 2000"
  $"03C1 F0FB 7878 0E38 1C3C 7807 0FFA 0012"
  $"0E30 0003 C1F0 F778 780E 381C 383F C73C"
  $"FA00 120E 1800 0381 E0F7 7070 1C38 3C38"
  $"3FE7 E0FA 0011 0D0C 0003 81E0 F670 701C"
  $"783C 3807 FFF9 0011 0D06 0000 03C1 ECF0"
  $"F038 7078 3800 FEF9 0011 0D03 0000 03C1"
  $"E8F0 E038 70F8 7800 7EF9 0011 0401 8007"
  $"03C1 FEE0 0570 71B8 70C0 7EF9 0012 0E00"
  $"C00F 8381 C0E0 E070 F338 71C0 6E03 FA00"
  $"120E 0060 0F87 83C1 EDE0 E0F6 3B77 C0CF"
  $"06FA 0012 0E00 300F 8783 C1F9 F1C0 FC7E"
  $"7CE3 8F8C FA00 120E 0018 0F87 0B81 F1FF"
  $"00F8 7C78 FF07 F8FA 0012 0E00 0C07 071B"
  $"80C1 FC00 6070 607C 03E0 FA00 0B07 0006"
  $"0000 3C00 03C0 F300 0B07 0003 0000 7800"
  $"03C0 F300 0B07 0001 8000 F000 03C0 F300"
  $"1715 0000 C001 E000 03C1 F1F0 78E7 1CE0"
  $"F0E0 3C0F 1C7C 3C00 1715 0000 6003 C000"
  $"0381 5090 8842 0C41 1840 4611 0824 4400"
  $"1715 0000 3007 8000 0780 4081 0042 0A42"
  $"0840 8220 0820 4000 1715 0000 180F 0000"
  $"0780 40E1 007E 0A42 0840 8220 0838 3800"
  $"1715 0000 0C1E 0000 0780 4081 0042 0942"
  $"0840 8223 8820 0C00 1715 0000 063C 0000"
  $"0780 4081 0042 0942 0840 8221 0820 0400"
  $"1603 0000 0378 FD00 0D40 9188 4208 C310"
  $"44C4 3108 2444 0016 0300 0001 F0FD 000D"
  $"E1F0 F0E7 1CC1 E0FC 781E 1C7C 7800 06FE"
  $"0000 E0EF 0006 FE00 0040 EF00 A000 83FF"
};

resource 'DLOG' (128, preload) {
  (0, 0, 160, 418),
  dBoxProc,
  invisible,
  noGoAway,
  0x0,
  128,
  ""
};

resource 'DITL' (128, preload) {
  {
    {10, 10, 42, 42},
    Icon {
      enabled,
      128
    },
    {14, 52, 30, 408},
    StaticText {
      enabled,
      "TearOffPalette by Don Melton and Mike Ritter"
    },
    {30, 52, 46, 408},
    StaticText {
      enabled,
      "©1987, 1988 by Impulse Technologies, Inc."
    },
    {56, 10, 72, 408},
    StaticText {
      enabled,
      "Designed for MacTutor: The Macintosh "
      "Programming Journal"
    },
    {72, 10, 88, 408},
    StaticText {
      enabled,
      "Compiled with LightspeedC™ version 2.15"
    },
    {98, 190, 130, 408},
    StaticText {
      enabled,
      ""Sure. We can do that. We don't even have to "
      "have a reason.""
    },
    {134, 190, 150, 408},
    StaticText {
      enabled,
      "Call us at (408) 296-6110"
    },
    {98, 10, 149, 176},
    Picture {
      enabled,
      129
    }
  }
};

resource 'TOPD' (0) {
  "TearOffPalette by Don Melton and Mike Ritter, "
  "©1987, 1988 by Impulse Technologies, Inc., "
  "all rights reserved."
};

data 'TOMG' (131, preload) {
  $"0000 0000 0000 0000 0000 0000 0000"
  $"0000 0000 0000 0000 0000 0000 0000"
};
```


© The Definitive MacTutor, Vol. 4

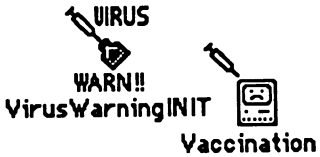
```
data 'TOMG' (132, preload) {
   $"0000 0000 0000 0000 0000 0000 0000"
   $"0000 0000 0000 0000 0000 0000 0000"
};

data 'TOMG' (133, preload) {
   $"0000 0000 0000 0000 0000 0000 0000"
   $"0000 0000 0000 0000 0000 0000 0000"
};

resource 'MENU' (128, preload) {
   128,
   textMenuProc,
   0x7FFFFFFD,
   enabled,
   apple,
   {
      "About TearOffPalette…", noIcon, "", "", plain,
      "-", noIcon, "", "", plain
   }
};

resource 'MENU' (129, preload) {
   129,
   textMenuProc,
   0x7FFFFFF5,
   enabled,
   "File",
   {
      "New", noIcon, "N", "", plain,
      "-", noIcon, "", "", plain,
      "Close", noIcon, "W", "", plain,
      "-", noIcon, "", "", plain,
      "Quit", noIcon, "Q", "", plain
   }
};

resource 'MENU' (130, preload) {
   130,
   textMenuProc,
   0x7FFFFFFD,
   enabled,
   "Edit",
   {
      "Undo", noIcon, "Z", "", plain,
      "-", noIcon, "", "", plain,
      "Cut", noIcon, "X", "", plain,
      "Copy", noIcon, "C", "", plain,
      "Paste", noIcon, "V", "", plain,
      "Clear", noIcon, "", "", plain
   }
};

resource 'MENU' (131, preload) {
   131,
   128,
   allEnabled,
   enabled,
   "Tool",
   {  /* 16 elements */
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain
   }
};

resource 'MENU' (132, preload) {
   132,
   128,
   allEnabled,
   enabled,
   "Pattern",
   {  /* 64 elements */
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
```

```
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain
  }
};

resource 'MENU' (133, preload) {
  133,
  128,
  allEnabled,
  enabled,
  "Color",
  {  /* 256 elements */
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
    "*", noIcon, "", "", plain,
```

```
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain,
      "*", noIcon, "", "", plain
  }
};

resource 'SIZE' (-1) {
  dontSaveScreen,
  acceptSuspendResumeEvents,
  doOwnActivate,
  393216,
  393216
};

resource 'SIZE' (Ø) {
  dontSaveScreen,
  acceptSuspendResumeEvents,
  doOwnActivate,
  393216,
  393216
};

resource 'ICON' (128, purgeable) {
  $"FFFF FFEØ 8ØØØ ØØ3Ø 8ØØØ ØØ3Ø EFBF FFFØ"
  $"888Ø ØØ3Ø E8BF FFFØ 888Ø ØØ3Ø EFBF FFFØ"
  $"8ØØØ ØØ3Ø 81FF FFFE FFØØ ØØØ2 81BA AAAA"
  $"8128 ØØØ3 81BA AAAB 81ØØ ØØØ3 81FF FFFF"
  $"81ØØ ØØØ3 81ØØ ØØØ3 81ØØ ØØØ3 81ØØ ØØØ3"
  $"81ØØ ØØØ3 81ØØ ØØØ3 81ØØ ØØØ3 81ØØ ØØØ3"
  $"81ØØ ØØØ3 81ØØ ØØØ3 81FF FFFF 8Ø3F FFFF"
  $"8ØØØ ØØ3Ø 8ØØØ ØØ3Ø FFFF FFFØ 7FFF FFFØ"
};
```

Advanced Mac'ing

A Vaccine for the 'nVIR' Virus



Mike™ Scanlin
Contributing Editor
MacTutor Vol. 4 No. 5

Unless you are going to Africa or Indochina, viruses and vaccinations are not something that most of us need to worry about. However, even if you're not planning on travelling, there is one virus you need to be aware of. It is a computer virus that is infecting Macintoshes everywhere. *[Note: The virus described in this article is apparently only one of at least three viruses that are going around as reported in the press. This article discusses what we shall name the 'nVIR' virus. The other two are the infamous 'peace message' virus and the 'scrapbook' virus reported in this month's mousehole column, and in a recent MacWeek article. -Ed]*

Are you infected?

Use ResEdit to open your system file and look for 'nVIR' resources. If you have them, then your system has been infected and chances are that at least some (if not most or all) of your applications are infected. Don't panic. This particular virus is relatively harmless. There is an application at the end of this article that will allow you to remove the virus from your infected applications. There is also an 'INIT' resource you can put in your System Folder that will warn you if this virus ever shows up on your system. *[Note that this vaccine and virus warning init applies only to this particular 'nVIR' virus. New vaccines will be necessary for the other two once it is known how they operate. -Ed]*

How I found it

Until last week, I had had no experience with computer viruses. I had heard rumors about the existence of Mac viruses, but didn't really believe them. I do not know when this virus first got into my system. It must have come from some program I downloaded off of a network, but I do not know which one. By the time I figured out what was going on, the virus had modified seventeen of the applications on my hard disk and my System file.

Virus Symptoms

Sometime near the beginning of last week, I started hearing a beep when launching programs. It didn't happen every time, only once in a while and with no discernable pattern. Using TMON, I trapped SysBeep() and discovered that something was modifying 'CODE' 0 and installing several 'nVIR' resources into every application I launched. I looked in my System file and, in addition to several 'nVIR' resources, found an 'INIT' 32 resource that I didn't put there. I compared the standard 'INIT's from an original system disk and none of them matched the 'INIT' 32 I had found. What really clued me in to the idea of a virus was that if I took the 'INIT' 32 resource out of my System file, quit ResEdit, and then relaunched ResEdit, the 'INIT' 32 resource would be back in there. After disassembling 'INIT' 32, I learned how it worked and how to make my system immune to it. I am sharing this information so that other Mac users can protect themselves as well. *[Note that this virus exhibits the ability to re-install itself after being patched out with ResEdit! -Ed]*

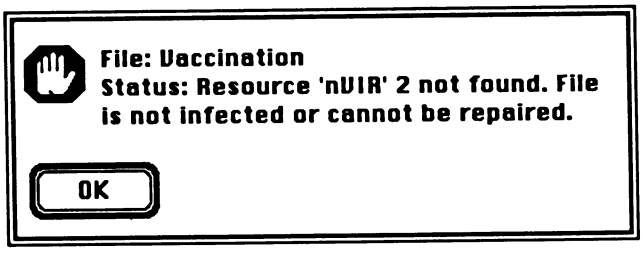


Fig. 1 Vaccination Alert tells Application status

How to make your System file immune

Use ResEdit to open your System file. Create an 'INIT' 32 resource that consists of these 2 hex bytes: 4E 75 (which is an RTS instruction). If 'INIT' 32 already exists and has a size of 366 bytes, then you can be pretty sure it is the virus' 'INIT'. Replace the existing 'INIT' 32 with the 2 byte version (4E 75). Now create 8 resources of the type 'nVIR'; the case of the resource type is important — do not use 'NVIR' or 'nvir'. Their IDs should be 0 through 7, with size zero bytes. If they already exist, then delete them and create 8 new empty ones (with IDs 0-7).

That's it. Your system is now immune to this particular virus (but not all possible viruses). If you now run an infected application, the virus will think that it is already installed in your system file, since it sees the 'INIT' and 'nVIR' resources it expects, and will leave it alone.

If your System file was infected before you immunized it, you should reboot the system before using the procedure below to remove the virus from your applications. This guarantees that the effects of 'INIT' 32 are removed from memory.

Removing the virus from infected applications

If an application has been infected, it will have several 'nVIR' resources, a 'CODE' 256 resource, and a possibly modified 'CODE' 0 resource. Here are instructions on how to restore an infected application (note: this is only useful if you are certain that your System file is not infected. Otherwise, the applications will become infected again. Also, you should practice on a copy of an infected application):

1) Open the application with ResEdit. If 'CODE' 256 exists, use GetInfo on it to check its size. If it is 372 bytes,

© The Definitive MacTutor, Vol. 4

then remove it. The reason we check for the size is because some applications, such as ReadySetGo, already have a 'CODE' 256 resource of their own and we don't want to remove part of the application's code.

2) Open 'CODE' 0 and look at the 3rd line of 8 hex bytes (bytes 16-23). If it is "0000 3F3C 0100 A9F0" then you need to replace that line of hex numbers with the 8 bytes contained in the 'nVIR' 2 resource. If the third line does not look like the above 8 bytes, then the 'CODE' resource is probably protected and did not get modified — see below for an explanation. In this case leave it alone.

3) Remove all 'nVIR' resources. Make sure you have completed step 2 before removing 'nVIR' 2. You cannot restore the application without it.

Because this procedure is so automatic, I have written a program that does it for you. The application Vaccination displays the SFGetFile dialog and allows you to choose an application to vaccinate. A message is displayed that tells you the result of the vaccination and the SFGetFile dialog is displayed again. If your system has been infected, you should vaccinate every application on your hard drive. You will only see files of type 'APPL', 'FNDR' (for the Finder), and 'dahd' (for the DA handler) in the SFGetFile dialog so you might want to do a manual tree walk of your hard drive to be sure you vaccinate all of your applications. There is no harm in vaccinating an uninfected application or in vaccinating the same application more than once. This program does not make applications immune to this virus, it only removes this virus from them. But if your System file is immune, then there is no way this particular virus can spread to your applications. Note: you cannot use the Vaccination program to make your System file immune. You will have to do that manually using the procedure above.

How this virus works

This particular virus modifies the 'CODE' 0 resource of an application in such a way that when you launch that application the first thing to execute is a piece of virus installation code. That installation code looks for the virus' presence in the System file you are launching from. If it does not find evidence of the virus, it then installs itself (as 'INIT' 32 and several 'nVIR' resources) into your System file and then executes the application you had originally launched. Once your System file is infected, every application launched from that system will become infected. The whole infection process only takes a second or two, so there is little chance you will notice it. If the virus detects that it is already in the System file and in the application you are launching (meaning that no installation of itself is necessary on this launch), then there is about a 6% chance (1 in 16) that you will hear a short beep. This is the beep that first got my attention. According to a friend of mine, Chris Borton, whose computer was also infected, if you have MacinTalk in your System Folder, then the virus speaks the words "Don't Panic" instead of beeping.

This virus does not check if the 'CODE' 0 resource of the application it is trying to infect is protected or not. Consequently, applications that have 'CODE' 0 resources with the resProtected bit set are still infected, but are not contagious, i.e. they have the 'CODE' 256 resource and the 'nVIR' resources added to them, but they can not pass the virus on to a clean System file. I learned this by noticing that QUED/M and PageMaker were infected, but were not contagious. I couldn't figure out why some programs had protected 'CODE' resources and others didn't. Then one of the people I work with, Victor Romano, put it together. He told me that Lightspeed C (which QUED/M and PageMaker were written in) automatically sets the resProtected bit of the 'CODE' resources it generates. MPW does not. So, protecting the 'CODE' resources (which can be done with ResEdit) is another simple way of preventing this virus from affecting an application.

To be forewarned

I don't know how far this virus has already spread, or how far it will spread. As a partial defense, however, I have written a piece of code that can be installed as an 'INIT' file in your System Folder that will warn you if it detects something that looks like this particular virus. VirusWarnINIT is a patch on 2 routines that this virus relies on: GetResource() and ChangedResource(). The patch to GetResource() makes a beep if theType == 'nVIR'. The patch to ChangedResource() makes a beep if theResource is a handle to a 'CODE' 0 resource. I wouldn't suggest installing this 'INIT' in a system known to be infected — the number of beeps is sure to annoy you. I would have used something like an alert window instead of a beep as a warning, but I can't be sure that the Window Manager has been initialized at the time the virus is detected. If you install this 'INIT' in a clean system and then launch a contagious application, you will hear about 5 or 6 beeps in a row as the virus tries to install itself in your System file.

Note that this 'INIT' is only a warning, not a vaccination. The virus will still install itself. The advantage is that you will know about it right away and can stop it before it spreads very far.

Now that my Mac has been vaccinated, it's my turn. After Typhoid, Yellow Fever, Cholera and Meningococcal vaccinations, I'm off to Africa and Indochina. I wonder if I can get David Smith to send MacTutor to Serengeti National Park? Or do they already get it there? I'll let you know...

```
/* Vaccination.c
   * by Mike™ Scanlin   12 March 88
   *
   * Removes the 'nVIR' virus from an
   * application chosen by the user.
  */

#include  "QuickDraw.h"
#include  "ResourceMgr.h"
#include  "StdFilePkg.h"
#include  "FileMgr.h"

#define   NIL       0L
#define   reg       register

#define   REPORT_STATUS_ALERT 129

#define   nVIR_CODE_256_SIZE  372

#define   nVIR2Bad            -10
#define   nVIR2NotFound       -11

void      RemoveResourceFromFile(long theType, int theID, int
   refNum);
```

```
int       Innoculate(Str255 *fileName, int vRef);
void      pStrCpy(char *p2, char *p1);
Boolean   ChooseFile(Str255 *fn, int *vRef);
void      main(void);

static SFReply   reply;
static int       applResFile;


/* RemoveResourceFromFile(theType, theID, refNum)
   *
   * This will remove the resource of type theType
   * and ID theID from the open resource file
   * whose refNum is refNum.
  */
void   RemoveResourceFromFile(theType, theID, refNum)
long   theType;
int    theID;
int    refNum;
{
   reg Handle    theResource;

   if ((theResource = GetResource(theType, theID)) &&
(HomeResFile(theResource) == refNum))
      RmveResource(theResource);
}

/* Innoculate(fileName, vRef)
  *
  * This removes the 'nVIR' virus from *fileName
      in directory vRef.
  */
int       Innoculate(fileName, vRef)
reg Str255    *fileName;
int           vRef;
{
   reg Handle       oldCODE, currentCODE;
   reg int          i, refNum, returnVal;
   ParamBlockRec    pb;

   /* init the ParamBlockRec to all zeros */
   asm {
         lea    pb,A0
         move.l #sizeof(ParamBlockRec),D0
         subq.l #1,D0

      @1 clr.b  (A0)+
         dbra   D0,@1
   }

   /* set the current working directory */
   pb.ioParam.ioVRefNum = vRef;
   PBHSetVol(&pb, FALSE);

   refNum = OpenResFile(fileName);

   if ((oldCODE = GetResource('nVIR', 2)) &&
(HomeResFile(oldCODE) == refNum)) {
      if (GetHandleSize(oldCODE) != 8)
         /* if 'nVIR' 2 isn't 8 bytes, then something
            isn't right. */
         returnVal = nVIR2Bad;
      else {
         if ((currentCODE = GetResource('CODE', 0)) &&
(HomeResFile(currentCODE) == refNum)) {
            asm {
                  MOVE.L oldCODE, A1
                  MOVE.L (A1),A1
                  MOVE.L currentCODE, A0
                  MOVE.L (A0),A0
                  ADDA   #16, A0
                  MOVE.L (A1)+, (A0)+
                  MOVE.L (A1), (A0)
            }
            ChangedResource(currentCODE);
         }

         /* kill the 'nVIR' resources */
         for (i = 0; i <= 7; i++)
            RemoveResourceFromFile('nVIR', i, refNum);

         /* kill the extra 'CODE' resource that this
               virus adds (only if it has the size
               of nVIR_CODE_256_SIZE) */
         if ((currentCODE = GetResource('CODE', 256)) &&
            (GetHandleSize(currentCODE) ==
            nVIR_CODE_256_SIZE) && (HomeResFile(currentCODE)
            == refNum))
               RmveResource(currentCODE);

         returnVal = noErr;
      }
   }
   else
      returnVal = nVIR2NotFound;

   if (refNum != applResFile)
      CloseResFile(refNum);

   return(returnVal);
}

/* pStrCpy(p2, p1)
  *
  * Copy the pascal string at *p1 to *p2.
  */
void pStrCpy(p2, p1)
reg char  *p2, *p1;
{
   reg int   len;

   len = *p2++ = *p1++;
   while (-len >= 0)
      *p2++ = *p1++;
}


/* ChooseFile(fn, vRef)
   *
   * Use SFGetFile() to get the name of an
   * application from the user. Return the directory
   * of the chosen file in *vRef. Return FALSE if the
   * user clicked Cancel, TRUE if they clicked
   * Open.
   *
   * Thanks to Chris Borton for this routine.
   */
Boolean ChooseFile(fn, vRef)
Str255 *fn;
int    *vRef;
{
   SFTypeList myTypes;
   static Point SFGwhere = { 90, 82 };

   myTypes[0] = 'APPL';
   myTypes[1] = 'FNDR';
   myTypes[2] = 'dahd';

   SFGetFile(SFGwhere, NIL, 0L, 3, myTypes, 0L, &reply);
   if (reply.good) {
      pStrCpy((char *)fn , (char *)reply.fName);
      *vRef = reply.vRefNum;
      return(TRUE);
   }
   else
      return(FALSE);
}
```

```
void main() {

   Str255 fileName;
   int    vRef;

   /* save the application's resource file refNum */
   applResFile = CurMap;

   InitGraf(&thePort);
   InitFonts();
   InitWindows();
   InitMenus();
   InitDialogs(0L);
   InitCursor();

   /* keep choosing files until the user hits Cancel */
   while (ChooseFile(&fileName, &vRef)) {
      switch (Innoculate(&fileName, vRef)) {
      case  nVIR2Bad:
         ParamText(&fileName, "\pResource 'nVIR' 2 is not 8
bytes long. File cannot be repaired.", NIL, NIL);
         break;
      case nVIR2NotFound:
         ParamText(&fileName, "\pResource 'nVIR' 2 not found.
File is not infected or cannot be repaired.", NIL, NIL);
         break;
      default:
         ParamText(&fileName, "\pVirus successfully removed.",
NIL, NIL);
      }

      /* show the result of the attempted removal */
      Alert(REPORT_STATUS_ALERT, NIL);
   }
}
```

```
/* VirusWarnINIT.c
   * by Mike™ Scanlin   13 March 88
   *
   * Put this in your system folder to warn you
   * about the 'nVIR' virus.
   * It patches GetResource() and ChangedResource().
  */

#include "Asm.h"
#include "ResourceMgr.h"

#define   GetResource      0xA9A0
#define   ChangedResource  0xA9AA
#define   JMP              0x4EF9
#define   memFullErr       -108
#define   beepDuration     20

void   main(void);

void   main()
{
   asm {
/* beginning of the code that installs the patches */
            move.l D3,-(SP)
            /* get the original GetResource address */
            move   #GetResource,D0
            _GetTrapAddress
/* set up JMP instruction that calls original GetResource */
            lea    @origGR,A1
            move   #JMP,(A1)+
            move.l A0,(A1)
/* get the original ChangedResouce address */
            move   #ChangedResource,D0
            _GetTrapAddress
/* set up JMP instruction that calls orig ChangedResource */
            lea    @origCR,A1
            move   #JMP,(A1)+
            move.l A0,(A1)

/* get some space in the system heap for the patches */
            lea    @last,A0
            lea    @first,A1
            suba.l A1,A0     /* the length of our patches */
            move.l A0,D0
            add.l  #254,D0/*xtra space for Str255 (@name) */
            move.l D0,D3     /* save for _BlockMove */
            _NewPtr   SYS
            cmpi      #memFullErr,D0
            beq.s     @noPatch
            move.l    A0,-(SP)      /* save for _BlockMove */
            move.l    (SP),-(SP)    /* for _SetTrapAddress */

/* set GetResource to the beginning of the space we just got
in the system heap */
            move   #GetResource,D0
            _SetTrapAddress

/* set ChangedResource trap */
            lea    @changedResouce,A0
            lea    @getResource,A1
            suba.l A1,A0
            adda.l (SP)+,A0
            move   #ChangedResource,D0
            _SetTrapAddress

/* now move it into place */
            lea    @first,A0
            move.l (SP)+,A1
            move.l D3,D0
            _BlockMove

@noPatch    move.l (SP)+,D3
            rts

/* end of the code that installs the patches */

@first

/* * This is the patch to GetResource */
@getResource
            move.l 6(SP),D0  /* get theType */
            cmpi.l #'nVIR',D0
            bne.s  @origGR

/* at this point we know something is trying to load an 'nVIR'
resource */
            move   #beepDuration,-(SP)
            _SysBeep
/* note that this is only a warning beep, it falls through
   * to the original GetResource, so the calling function
   * never knows that it was detected. */
@origGR     nop          /* JMP to original trap */
            nop
            nop

/* * This is the patch to ChangedResource*/
@changedResouce
/* call GetResInfo() to see if changing a CODE 0 resource */
            move.l 4(SP),-(SP)  /* copy resource handle
                                   * that was passed to
                                   * ChangedResouce() */
            pea    @theID
            pea    @theType
            pea    @name
            _GetResInfo
            lea    @theType,A0
            move.l (A0),D0
            cmpi.l #'CODE',D0
            bne.s  @origCR

            lea    @theID,A0
            tst    (A0)
```

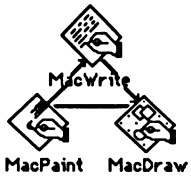
```
            bne.s  @origCR

/* give a warning beep a CODE 0 resource is being changed */

            move   #beepDuration,-(SP)
            _SysBeep

@origCR     nop          /* JMP to original trap */
            nop
            nop
@theID      dc     0
@theType    dc.l   0
@name       dc     0
         /* there are actually 254 more bytes to this
         * variable (for a total of 256). Check the
         * add.l #254,D0 instruction in the install code. */
@last
   }
}
```

```
* Vaccine.r Application
* by Mike Scanlin
*
vaccine.RSRC
????????

Type VACC = STR
   ,0
Vaccine for the 'nVIR' virus by Mike Scanlin

Type BNDL
  ,128 (0)
VACC 0
ICN#
0 128
FREF
0 128

Type FREF
  ,128 (0)
APPL 0

Type SIZE = GNRL
   ,0
.I
00000  ;; $4000 = bit 14 clear
.L
128000 ;; recomended
.L
128000 ;; minimum

Type SIZE = GNRL
   ,-1
.I
00000  ;; $4000 = bit 14 clear
.L
128000 ;; recomended
.L
128000 ;; minimum

Type ALRT
  ,129 (0)
80 64 188 448
129
4444

Type DITL
  ,129 (0)
3

Button
78 20 98 80
OK

IconItem Disabled
10 32 42 64
0

staticText Disabled
10 110 112 374
File: ^0 \0D++
Status: ^1

Type ICN# = GNRL
  ,128 (0)
.H
2000 0000 4000 0000 A800 0000 1400 0000
2200 0000 1100 0000 0880 0000 0440 0000
0220 0000 0120 0000 00E0 0000 0010 0000
0008 0000 0000 0000 0007 FFC0 0008 0020
0009 FF20 000A 00A0 000A 44A0 000A 00A0
000A 38A0 000A 44A0 000A 00A0 0009 FF20
0008 0020 0008 0020 0009 0F20 0008 0020
0008 0020 000A AAA0 0008 0020 000F FFE0
2000 0000 4000 0000 A800 0000 1C00 0000
3E00 0000 1F00 0000 0F80 0000 07C0 0000
03E0 0000 01E0 0000 00E0 0000 0010 0000
0008 0000 0000 0000 0007 FFC0 000F FFE0
000F FFE0 000F FFE0 000F FFE0 000F FFE0
000F FFE0 000F FFE0 000F FFE0 000F FFE0
000F FFE0 000F FFE0 000F FFE0 000F FFE0
000F FFE0 000F FFE0 000F FFE0 000F FFE0


* VacInit.r INIT resources
* for detecting 'nVIR' virus
* by Mike Scanlin
*
VacInit.RSRC
????????

Type VNIT = STR
   ,0
Virus detection init by Mike Scanlin

Type BNDL
  ,128 (0)
VNIT 0
ICN#
0 128
FREF
0 128

Type FREF
  ,128 (0)
INIT 0

Type ICN# = GNRL
  ,128 (0)
.H
2015 724E 4015 4A50 A815 4A50 1415 724C
2215 5242 1115 4A42 0889 499C 0440 0000
0220 0000 0120 0000 00E0 0000 0010 0000
000B 0000 0007 B000 000E F800 000C 0C00
0006 F600 0002 FB00 0006 F600 0006 EC00
0003 5800 0001 B000 0000 E000 0000 4000
0000 0000 1523 9128 1522 5928 1552 5528
1553 9528 0A72 9528 0A8A 5300 0A8A 5128
2015 724E 4015 7A50 A815 7A50 1C15 724C
3E15 5242 1F15 4A42 0F89 499C 07C0 0000
03E0 0000 01E0 0000 00E0 0000 0010 0000
000B 0000 0007 B000 000F F800 000F FC00
0007 FE00 0003 FF00 0007 FE00 0007 FC00
0003 F800 0001 F000 0000 E000 0000 4000
0000 0000 1523 9128 1523 D928 1573 D528
1573 9528 0A72 9528 0A8A 5300 0A8A 5128
```

© The Definitive MacTutor, Vol. 4

Advanced Mac'ing

Protocols for an IAC Driver, Part 1



Frank Alviani
Contributing Editor
Paul Snively
Contributing Editor
MacTutor Vol. 4 No. 8

How to Get Your Applications Talking to Each Other Again

The meeting had just broken up, Paul and I were at the local beer-and-beef joint, and we were feeling rather annoyed. We had been told that Apple wasn't going to be implementing any sort of method for *really* using MultiFinder to the max until at least System 7. No form of inter-application communications until at least then, they said.

"Can't really be all that difficult," I said over a cookie milk shake. "For Pete's sake, you could use a device driver to get the needed 'alien' memory needed for communicating! Drivers normally have to grab some working memory when they open anyway, so it's no big deal."

"Of course it's not that hard! You and I could probably do a complete job—with sample applications to show Dave Smith that we're not circling in a low orbit— in a few evenings," Paul rumbled back over his cookie shake (they make awfully good shakes there...). "In fact, we oughta do just that! Show Apple all the talent isn't inside the city limits of Cupertino..."

Thus was born the SAWS Inter-Application-Communications Protocol (detailed in the accompanying description of the protocol). Our purpose in writing it was to show that communicating between programs under MultiFinder isn't really that difficult in principle (a proof by demonstration) and have some fun in the process. In the next few months we will be publishing the device driver, its installation INIT, and a variety of samples. Various other members of the Chicago-area Mac Gang are joining in on the fun by writing and submitting simple applications such as graphing programs, text editors, etc. that communicate between each other using the IAC protocols.

While the protocol is copyrighted, anybody may use it freely.

We hope that encouraging its widespread use will help spread the concept of mix-and-match software in the community and let people discover how nice really personalized, "custom-blended" software is. Widespread experimentation in the freeware/shareware arena may also help us all find out what interface approaches work and what approaches don't.

Assumptions: I made the following assumptions about my users in designing the protocol:

1) Programs must explicitly be written to use this protocol. It is NOT written to be retro-fitted onto existing applications. Many developers grumble about MultiFinder and the numerous kludges it contains in order to be compatible with existing programs; the fact is, however, that MF is a triumph of programming precisely because it succeeds in that compatibility with programs written when it didn't exist. I am not convinced that a backwards-compatible approach is even possible.
2) They are using several programs together on a single machine, rather than across a network. This simplifies communications greatly.
3) Documents will tend to be used together on the same machine, most often the machine where they were created. This reduces the probability of "collisions" between documents created on different machines but being processed together. We can thus use a simpler method of identifying documents so that they can "recognize" each other.
4) Enough information must be stored with documents to allow links to be reestablished as automatically as possible when stored documents are reopened.
5) The user interface should seem to be a natural extension of the existing interface, needing minimal learning and not intruding into the normal operation of the programs using it. Larry Atkin at Odesta suggested presenting it as an extended clipboard, hence our "magic clipboard" metaphor.

Basic Concepts: Given 2 programs running together under MultiFinder, they are linked if changing a specified chunk of data in document 'A' (the source) results in the changed data being used in document 'B' (the target) without any explicit intervention by the user. We say that the target document is dependent on the source document. This idea is central to our entire design.

Notice that while we assumed 2 programs running under MultiFinder, we spoke of documents being dependent on each other. This is due to the ability of most programs to manage multiple documents; the programs carry out the communication, but the documents are related.

There are two mechanisms needed for programs to be able to communicate automatically under MultiFinder (or any multitasking system, for that matter):

1) A method for a "target" program to determine that a chunk of data it is interested in has changed, so that it can retrieve that data.
2) A method of transmitting chunks of data between programs.

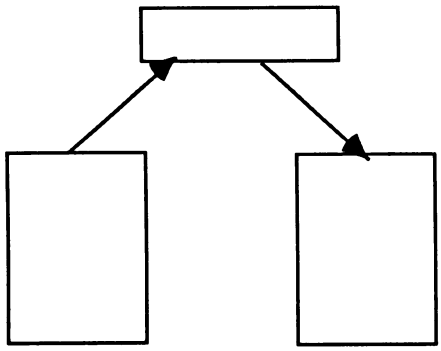
Our protocol addresses both of those problems in a simple fashion. Let me emphasize that there are many ways of addressing them—the designer's choice of methods depends on the needs of her/his users. Apple's final approach will probably be quite different from ours (but who knows)?

Keeping track of changed data: This is the feature of the IAC protocol that allows data to flow between programs without user intervention. Since there are 3 parties involved with the process of getting data between programs (source, driver, and

target),
there are 3 places where knowledge of the relationship between programs can be kept. Having the IAC driver keep track of which target documents are interested in a given chunk of data (called an "extent" in the protocol description) is the simplest solution:

1) Source programs simply publish the latest version of an extent when it has changed. They have no knowledge of who's interested, thus keeping their lives simpler.
2) A target program can retrieve data from a number of source programs with calls to just the IAC driver, rather than having to have some way of communicating with each source program explicitly.

Since an extent of changed data must be stored by the IAC driver anyway, we have added a feature to deal with the possibility of data being updated before all target documents have retrieved it. This is called an "edition," and starts with 1 when data is first written to the driver. If a new edition of an extent is written to the driver before all target documents have retrieved an



existing version, the latest edition is linked at the end of a chain. Target documents, by convention, remember the last edition read and simply request the latest edition higher than that.

When a document makes a data extent available to other programs for the first time, it is issued a document ID by the IAC driver, which serves as part of an "address" in communicating later. A properly written IAC application will store the document ID for each extent it is a target of so that (in a later session) should that document be opened the links can be automatically re-established.

Communicating between programs: The IAC driver serves in some respects as a set of highly intelligent clipboards, or mailboxes. A source program actually writes a chunk of changed data to the driver, using normal write calls to the file system, where it is stored until all the target programs that are interested in that chunk have read it, using normal read calls to the file system. While temporarily storing the data in the driver does take space, our only alternative would be to manipulate the data in different application heaps. This would require too much knowledge about the unpublished guts of MultiFinder to be either stable or safe.

A major implication of using normal file system calls for communicating is that target programs must actively poll the IAC driver in order to determine if the data they need has changed. This can be done both in the foreground and the background. However, if the target application genuinely needs to ignore the outside world for a while, there is no penalty since succeeding editions of each data extent are maintained by the driver.

Since our driver has functions significantly different from those of a standard driver, it needs a different set of parameters. These are transmitted in the I/O buffer that is used in read & write calls; since control calls to drivers don't use a buffer, control calls aren't used.

Interfacing to the IAC driver: While the attached protocol description gives the details of how the driver works and how to work with it, there is no need for everybody to re-invent the interface. Therefore, in the sample test program that will accompany the actual driver, a set of interfaces in C is provided as a package that can be used "off the shelf." The interfaces deal with the messy details of setting up parameter blocks, doing the calls, and checking operation statuses. Since most of the intelligence resides in the driver, they are straightforward.

Without further ado, here is the definition of the protocol and its associated routines:

A Protocol for Inter-Application Communications Under MultiFinder

— Version 1.2 —

© by Frank Alviani and Paul Snively, June 1988
Permission granted to MacTutor for Publication
All Rights Reserved

This document is intended to define how programs may be written to support "hot links" between several programs executing en suite under MultiFinder, using a device driver to provide the communications facilities required. It is divided into the following sections:

1) Definition of the concepts implemented. In particular, "hot links" and "data dependency" are defined.
2) Definition of the facilities made available by the device driver, and how they are used by the "IAC-friendly" program.
3) Operation of the driver and its internal data structures.
4) Suggestions for user-interface implications of this protocol, and how they might be implemented.
5) A list of the data formats supported in this version.

It is our hope that these protocols will be adapted freely and further improve the power and sheer *fun* of the Macintosh interface.

Section 1—Definitions and Assumptions

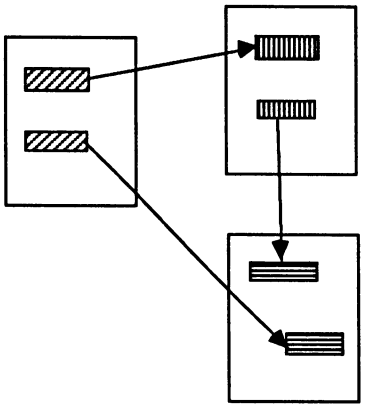
- An "extent" is the amount of data involved in a "data dependency". The source program must maintain the start and end of the extent in order to be able to detect when it has changed, so a change notice can be posted to all interested programs.
- A "data dependency" exists between a "source" document

and a "target" document when a change in an extent manipulated by the source program requires the data manipulated by the target program to change to correspond. In this protocol, the source program uses the device driver to make the changed data available to all interested target programs, and to post notification that the change has occurred. Target programs, in this protocol, are required to periodically poll the device driver to determine the latest state of the "foreign" data they are interested in.

It is vital to note that the basic objects dealt with are links between documents. This is required because most well-designed programs can manipulate several documents simultaneously. However, we often refer to source and target programs, since those are the machinery behind those documents; this is done when we are dealing with actually reading and writing the changing data.

Data dependencies involve several components. The complete set of data required must include:

1) The data itself, referred to as an extent.. The source program must maintain some way to remember its range - a block of cells in a spreadsheet, or a selection in a text document - so that it can be communicated after a change has occurred.
2) When a dependency is established by a source program, the IAC driver must have a way of identifying it later. This identification is in 2 parts:
   A) It issues a document ID. This is the clock time in seconds, assigned by the IAC driver when a dependency is first registered. We work under the assumption that it is impossible to complete all the actions required to establish and complete a dependency in less than 1 second and register a second dependency. Just in case someone is using Tempo, the driver can maintain a record of the last document ID issued and if the next one would be redundant, add 1 to maintain uniqueness.
   B) It issues a "hatcheck" for the extent - an index of the extents registered so far for that document. Extents are always identified by the combination of document ID and hatchecks, by both source and target programs. Hatchecks are unique and associated with a specific extent and document.
3) The identity of all target documents. A data source can be linked to up to n other documents. As a part of the updating process it is necessary to have some way for target programs to indicate that they've gotten the newest version of the data. The "slot_ID" is used for this purpose; it is an index into the IAC driver's internal table of registered documents.
4) An "edition" indicator for the data. When a piece of data is linked to a target document, it is given level 0; the 1st change results in level 1, etc. When several links exist between a source and a target document, the edition associated with each extent allows the target program to identify and "grab" just the data units it requires to be "up to date." When an existing document is opened all existing extents are assigned edition 0 to keep the bookkeeping manageable.
5) A format indicator. This serves the same purpose as it does for the clipboard. If a target program is unable to interpret any of the formats provided by the source program the dependency cannot be established and the user must be informed.



- A "hot link" is the user-interface for these data dependencies. It is the visual means of:

a) Specifying the range of source data to be echoed from the source program to the target program.
b) Indicating the location in the target document where the source data is to be included and shown.

- "External link definition" is the information stored in both the source and target documents that allows the data dependency to be re-established after they have been closed and re-opened and are being manipulated together.

Assumptions: The design laid out in this document is based on the following assumptions:

1) This is aimed primarily at individual users, or people working in small groups; this provides a simpler environment to plan for. In particular, this allows the use of fairly simple external link information.
2) Manual intervention by the user at any point should be absolutely minimal and transparent.
3) There is no need for "system administration" operations, and therefore are no provisions for them.
4) Programs need to be written to be "IAC" friendly; this is not a facility designed to be retrofitted to existing works.

Section 2 - Device Driver Facilities and their Usage

A) The device driver will be opened by an INIT31 resource at boot-up. It will remain open at all times. It will be named ".IAC." User programs gain access to it by opening it (which is harmless) in order to get an ioRefNum for further access. While we are less than thrilled at forcing the user

programs to open a driver with a specific and fixed name (it seems to be a rather "brittle" approach) there doesn't seem to be a better way.

B) Since the normal device driver calls do not have provisions for extensions to the standard parameter blocks, the necessary additional information will be passed at the beginning of the buffers that are part of the driver calls. That is, the buffers will consist of an "IAC parameter block" followed by one or more data blocks.

C) The basic operations supported by the driver are the following:

1) **Establish Dependency.** This is a 2-part process that involves both the source and the target programs. After the user has determined the data of interest, the source program tells the driver to "add this extent to the dependencies table," at which time it becomes the "available dependency." After the user has determined where the source extent is to be linked into the target document, the target program tells the driver "I am interested in the available dependency." Note that a benefit of the above approach is that several target programs can each indicate they are interested in the same source extent without having to re-establish it each time. A variation of this occurs when an open target document determines that it can re-establish an existing link with a source program, in which case the target program will specify the document and extent it is interested in.

2) **Remove Dependency.** This can be done by either the source or the target program. Target programs normally sever their dependencies when closing a document. If all target programs sever a dependency, the source program is notified so it can avoid the overhead of posting future changes; this notification occurs in the ioResult field of the next "UpdateData" or "Check status" calls.

3) **Identify a dependency as the "available dependency."** This is implemented to simplify the process of linking more than one target to a single source extent.

4) **Census.** This can be called to get a list of all registered extents, so that programs can determine if external links have changed. It would normally be called if the latest "check status" call showed that the number of registered documents had changed since the previous status call.

5) **Check status.** This call is normally made by all programs during their "null event" processing, in order to determine what can be done. If several links are being maintained, this can save the effort of making a "Read-Data" call for each extent. The user program should be very careful to make any needed calls in response to the IAC status before using any of the event-manager calls that could result in your program being swapped out; this will ensure the consistency of the environment.

6) **Update data.** The source program "writes" an extent to the IAC driver (identifying the extent, of course). The driver will assign it the next edition for that extent. It will be placed into a queue if any previous editions for that extent remain "unvisited" by any interested target programs.

7) **Read Data.** This is of the form "I am interested in extent 'x.' If it is more recent than edition 'y' please give me the new data." If the indicated extent is not newer 0 bytes of data will be returned.

D) It is important to note that the source program must copy the extent to the IAC driver, and the target program must copy it from the driver. This approach was chosen deliberately, in spite of the seeming overhead, for the following reasons:

1) It avoids the potentially severe memory-management problems associated with multiple heap-zones. Frankly, we don't know all the possible evil ramifications of trying to resize a relocatable block in another zone (for example) and don't care to find out the hard way.

2) The target document is guaranteed to have data available at all times, even when it is re-opened without the source document. Having a graph vanish from a text document because you forgot to open the spreadsheet the graphing program was dependent upon would be justly perceived as a Bad Thing.

E) The driver will not permit dependencies to be defined in the following situations:

1) In both directions between 1 extent in each of 2 documents. This is to avoid an infinite-loop of updates.
2) Within the same document.

External Link Information: This is crucial in allowing documents to re-establish links after they have been closed and are being reopened, without manual intervention. The basic principal is that programs must retain identifying information with documents that allows a "target" document to recognize its "source" documents; this information is given to the IAC driver when extents are first registered.

While the document_ID/hatCheck combination is not perfect (primarily in a larger environment where several documents could be initially registered simultaneously to within 1 second), it seems workable as the external link information and has the virtue of simplicity. While "collisions" are possible, we assume related documents would be exchanged together and therefore processed together, maintaining the original context.

The syntax of the IAC driver calls: These will be described in terms of the "standard" calls made to the device driver and the fields in the parameter block that starts the I/O buffer. All calls return a value in the ioResult field of the standard parameter block, which should be checked by the user program. Since the control message to I/O drivers makes no provisions for buffers, we need to use just the read/write calls.

In the parameter blocks below, links are identified by both "slot_ID" (for the program) and "hatCheck" (for the document/link). While the use of 2 fields may seem slightly redundant, it seems easiest to work this way, since there are occasions when we need identification of just the program.

1) **AddDependency:** via FSWrite()

```
block:  short   func = 1;     /* IN */
        long    docID;        /* IN/OUT */
```

```
short   slot_ID;     /* OUT */
short   hatCheck;    /* IN/OUT */
short   edition;     /* OUT */
```

docID is needed to identify the source document. If docID is zero, the source program has never registered this document with the IAC driver before; the assigned docID will be returned. docID should be stored with the document. As it identifies when the dependency is first registered (in seconds), it is extremely likely to be unique. docID should be 0 only for the first extent being registered for a document; all succeeding extents must have a valid docID value in order to be associated with the correct document.

slot_ID identifies the document during the course of the current session. It will be assigned by the IAC driver if 0 (zero) on input. If it is known (because the document is the target for 1 or more dependencies) it should be filled in to keep the internal tables of the IAC driver consistent.

hatCheck is the opaque identifier assigned to this extent. If this field is 0 (zero) on input, a value will be assigned by the IAC driver. When a valid docID and hatCheck are provided the driver assumes an existing extent is being registered by the source program. It must be used in all future communications regarding it, by all programs interested in it, both source and target(s).

edition will be returned as 0 (zero) in this version.

It will be necessary to use the "WriteData" call after the dependency is started to actually post the data to the driver.

2) **CompleteDependency:** via FSWrite()

```
block:  short   func = 2;    /* IN */
        long    docID;       /* IN/OUT */
        short   slot_ID;     /* IN/OUT */
        short   hatCheck;    /* IN/OUT */
```

docID is used to identify the source document. If it is 0 (zero) on input the IAC driver will assume the "available dependency" (this is a "defaulted" call); the hatCheck field will be ignored in that case. If it is non-zero both docID and hatCheck must match a registered extent.

slot_ID is used by the IAC driver to know what documents are targets for a dependency in order to maintain the "interest mask" (described below). The source document for the dependency, however, does not need to know and is never informed of the targets. As explained above, it should be filled in if known; if 0 (zero) the value will be assigned by the driver and must be provided in all future calls .

hatCheck is returned to identify the extent involved in the dependency (it is identical to the identifier issued by AddDependency to the source program), and must be used in all future communications with the driver.

The "ReadData" call must be made to actually retrieve the data for the newly established link.

3) **RemoveDependency:** via FSWrite()

```
block:  short   func = 3;    /* IN */
        long    docID;       /* IN */
        short   slot_ID;     /* IN */
        short   hatCheck;    /* IN */
```

docID and hatCheck are used to identify the extent and source document involved.

slot_ID is used to identify who is terminating their interest in the dependency.

If the source program is involved, the data for all editions of the extent are immediately purged. The edition will be set to -1 so that all interested target programs will be informed the next time they execute a "Status" or "ReadData" call. As target programs attempt to retrieve the latest version of the data they will be informed that the source has terminated the dependency, and the target's bit will be turned off in the interest_mask in the extent entry. When no interested targets remain, the extent entry is purged. If, instead, all target programs terminate their dependency on an extent, the source program is informed so that it can stop tracking changes in that extent.

4) **AvailableDependency:** via FSWrite()

```
block:  short   func = 4;    /* IN */
        long    docID;       /* IN */
        short   hatCheck;    /* IN */
```

This sets the indicated extent as the "available dependency" that will be the source to future defaulted "CompleteDependency" calls.

5) **Status:** via FSRead()

```
block: short   func = 5;     /* IN */
       short   slot_ID;      /* IN */
       short   vers_ID;      /* OUT - driver version */
       short   doc_count;    /* OUT - # active */
       short   extent_count; /* OUT - relevant */
```

slot_ID is passed in to identify the document that is making the inquiry.

vers_ID is always returned. It is an integer with an implicit scaling of 100; i.e. version 1.0 is returned as 100, version 1.01 as 101, etc.

doc_count is the number of documents currently registered.

extent_count is the number of extents relevant to the document.

6) **Census:** via FSRead()

```
block:  short   func = 6;          /* IN */
        short   extent_count;      /* OUT */
        /* for each extent: */
        long    docID;             /* OUT */
        short   hatCheck;          /* OUT */
```

extent_count is the total number of entries in the census.

each census entry consists of the current values for the docID and hatCheck fields.

7) **WriteData:** via FSWrite()

```
block:  short   func = 7;      /* IN */
        long    docID;         /* IN */
        short   hatCheck;      /* IN */
        short   edition;       /* OUT */
        short   format_count;  /* IN - # of formats */
        handle  the_data;      /* IN - handle to data */
/* In the block whose handle is passed, the following
* is repeated once per format supported by the user
* program
*/
        short   format_code;   /* format of this data */
        long    data_size;     /* size of this data */
```

```
..... the actual data .....
```

docID and hatCheck identifies the extent to be updated.

edition is returned for the use of the user program. It is optional whether it is retained with the document, but may be useful somehow.

format_count is the number of copies of the data that are being written in this updating of the extent. It can vary from update to update. For each copy of the data included in the block whose handle is passed in the buffer you must precede the data with:

format_code, as shown in the list later in this document.

data_size must be rounded up to an even number of bytes.

Now comes the actual data.

```
8) ReadData: via FSRead()
   block:  short    func = 8;          /* IN */
           long     docID;             /* IN */
           short    slot_ID;           /* IN */
           short    hatCheck;          /* IN */
           short    edition;           /* IN-OUT */
           short    format_pref[3];    /* IN */
    /* this is an array of preferred formats */
           short    format_code;       /* OUT */
           handle   the_data;          /* IN */
```

docID and hatCheck specifies the extent and source document.

slot_ID specifies the target document that is interested in it. edition is set by the user to the minimum change-level he is interested in (normally 1 more than the last edition it accessed); on output it contains the actual edition returned. The driver can (and probably will) skip any intervening editions, and mark them as "read." The user program should remember this value, but it need not be preserved when the program quits, since all extents start at level 0 when an existing document is opened.

format_pref is an array that specifies up to 3 data formats the user program can accept, in order of desirability. A null entry ends the list if there are less than 3 entries.

format_code will specify the actual format returned.

the_data is a handle to a block in which the data will be placed. It can be allocated to 0 bytes since the driver will resize the block as needed.

Section 3 - Operation of the IAC driver

- The driver will maintain a list of all currently-registered documents. This is a dynamic list. A fixed array of n (numbered 0 to n-1) entries is maintained. When a document terminates, it's slot is made available – the array is not compacted. The slot index is also the bit index in the "who's interested" word maintained with each extent.

When an extent is registered or linked to an additional document, the bits for each interested target document are turned on in the "interest mask". A copy of this is issued to each change-level when it is accepted by the driver. As a target program copies that change-level, its bit is turned off in the mask; when the mask hits 0, all interested parties have accessed it and it is discarded.

- The entry for an extent (not an edition!) is structured as follows:

```
handle   next_extent;          /* link */
long     document_ID;
short    hatCheck;
long     master_interest_mask;
handle   head_of_edition_list;
```

- The entry for an edition is structured as follows:

```
handle                  link_to_next_CL;
long                    interest_mask;
short                   number_formats_stored;
struct   (              /* 1 entry/format */
         short          format_code;
         handle         data_handle;
);
```

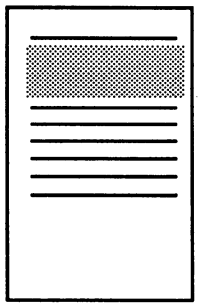
- The process of determining the dependencies that are of interest to a given document (in response to a Check_status call) is quite simple. The slot_ID of the inquirer specifies the bit to check in the interest mask for each active extent/edition, so it involves merely scanning every active extent/edition and remembering which ones have that bit set.

Section 4 - User Interface Suggestions

1) The most natural metaphor for the user is that of the "magic clipboard" that handles all the details of cutting and pasting for him/her automatically. In line with this metaphor, the establish-dependency item on the Edit Menu could well be called "Hot Copy," with the complete-dependency item called "Hot Paste." (Thanks to Larry Atkin for the suggested names.)
2) There should be a mechanism for identifying (displaying) active dependencies in a document. This can be implemented by some alternative highlighting of the extent, for example.
3) Clicking (double-clicking?) on a displayed dependency should cause the program (if it is the source program) to send a message to the IAC driver identifying that extent as the "available dependency" so it can be simply linked to a target document. This approach avoids problems associated with trying to link the same source extent to several documents (such as accidentally changing the range of data involved for different links). It may be desirable to have some slightly different highlighting (such as a darker stipple) in order to identify the "available dependency."
4) It is usually (but not always) desirable to post an update for an extent only when the changes are done and the data will be stable for a reasonable amount of time. In a text selection, for example, this means that you shouldn't post a data change for every character typed, but wait until the insertion point is placed outside the data range. There are times when every single change—no matter how small—should be posted immediately, so this is a somewhat elastic guideline.
5) In data structures that can be expanded & contracted (such as a text selection) the definition of the extent should be modified as needed as changes are made. In a text selection, for example, if some characters were inserted into the middle the original starting and ending characters would still be the start and end.

Section 5 - Data Formats Supported

1) Plain text.
2) Formatted text (a TE scrap record with style information).



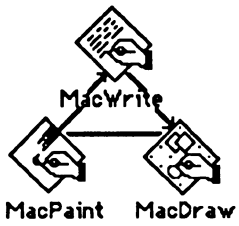
Stipple-Highlighted
text-document
link

3) PICT. Probably should include PICT2.
4) Spreadsheet range.

This is what we have designed. It will probably take some time for these concepts to soak in, which is just as well, since the driver (as of this writing) is still in the process of being implemented. Next month we will publish the commented MPW assembler source code of the driver, along with support stuff (like Rez source for its resources, an accompanying article, etc.) and our first sample application that supports the driver. Hopefully this has whetted your appetite somewhat, and, like us you can't wait to see a spreadsheet, graph program, and word processor, all running under MultiFinder, and to make a change to the spreadsheet, which then causes a change in the graph, which then causes a change in the report in the word processor. Until then, try to absorb this hodge-podge of information that Frank has put together and I have tried (ahem) to clarify and verify in terms of accuracy. Happy hacking!

Advanced Mac'ing

IAC Driver, Part II



Frank Alviani
Contributing Editor
Paul Snively
Contributing Editor
MacTutor Vol.4 No. 10

Inter-Application Communication Under MutiFinder:
The Continuing Saga,
or
Data, Data, Who's Got the Data?
or
If One Application is the Master
and the Other is the Slave,
Is the IAC Driver a Slave Driver?

Greetings, ladies and gentlemen, and welcome to yet another episode in the latest Macintosh development soap opera. In this segment, we ask the crucial question, "What does it actually take to make this puppy fly?" Luckily, Frank has spent a great deal of time and effort answering that question to his own satisfaction while I was out dancing with Shellie and Alice in Boston. (We should all be so lucky, I might add.)

Seriously, here is our attempt at cleaning up some of the mysteries surrounding how this driver we've written actually works. The definitive answer to that whole question can, of course, only be ascertained by careful examination of the source code, a herculean task at best.

Rather than put you, gentle reader, through that, here is our distillation of both what we wrote last month, and what is in the source code that accompanies this article (which is supposed to be reasonably short, since the code is far larger than anything MacTutor has ever published before).

Disclaimer!! This version of the IAC Driver is ß1!! We have had enough requests and inquiries to convince us to go ahead and release this early Beta version for people to experiment with, understanding that revisions are inevitable. THIS DRIVER UNDOUBTEDLY STILL HAS A FEW BUGS IN IT! Don't commit all your eggs to the IAC basket just yet!!

First of all, the driver had to be remarkably transparent (read: drop it in your System Folder and forget that it exists). For that reason it is a Startup Document (for you pre-System 6.0 freaks, it's an INIT file). Basically there's a ridiculously simple little INIT in there that opens the DRVR in the system heap, makes sure it's in the unit table, and then makes sure that it won't go away when the INIT file is closed (by detaching the resource). Peter Helme of Mac DTS kindly provided a small chunk of code that finds the highest unoccupied slot in the unit table and renumbers the DRVR resource to ensure it is loaded there.

Once the driver is there, applications are free to check to see if it's around and, if it is, to make use of it. The calls were documented last time as to name and function (sort of), so I'm going to try to focus on some of the not-so-obvious aspects of what goes on, and why (Frank is actually explaining the "why"; it's his design, after all).

First of all, understand one important fact: inter-application communication under MultiFinder (or any other multitasking environment, for that matter) is neither more nor less than an exercise in memory management. The source application has to allocate storage for the transfer of data to the target application, and somehow the target application has to be made aware that the block of memory has something that it wants. Also, since the source data can be changed in a fairly dynamic manner relative to the target retrieving the data, you have to keep track of various versions or "editions" of the data somehow. All of this is the driver's responsibility. It allocates and releases the storage (yes, Paul Sydney, it "does" release the storage), and it tracks editions as necessary.

A second matter to consider during program design is that inter-application communication is very much like network communication (only without the messy wires) and therefore similar problems arise, such as timing and coordination difficulties. If you've never dealt with those before, it's going to prove very stressful for a while (although the problems aren't really of the same magnitude).

Specification changes

Since the time the protocol was published in the August 1988 issue of MacTutor, a few small changes were found to be necessary in the specifications. Here they are:

1) The identifier for data types was changed from a 16-bit integer to the 4-character resource type used by everybody else (no, I don't recall the good reason I had for choosing an integer in the first place).

2) The information returned by the Census call for an extent now includes the latest edition, since that is necessary for the application to be able to determine if it needs to read that extent or not.

3) When an extent is deleted by its source, the entry is removed from the table entirely (rather than just setting the edition to -1 as originally stated). Targets that try to access that extent in the future will get a "No such extent" error from the driver, and will know to not try to read it anymore (the extent should NOT be purged from the application's internal tables, since re-opening the document in the future will re-establish the dependency).

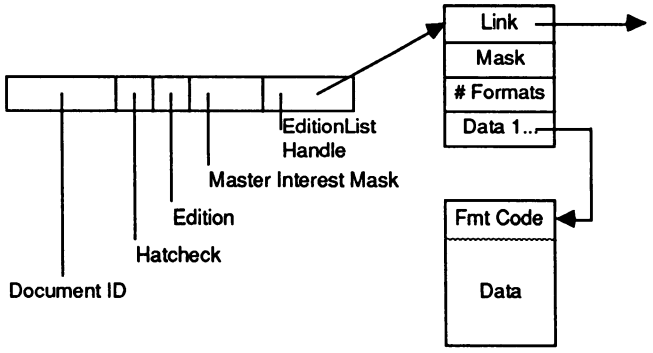
Internal Data Structures

One of the keys to understanding any piece of code is a good grasp of the data structures it uses: how they are used and why they were chosen. The data structures used in the IAC driver are not terribly complex.

The extent table – is the heart of the driver. It is a fixed-length

© The Definitive MacTutor, Vol. 4

table that includes information about where an extent originates, who is interested in it, and what is the latest version of data that each target has read.



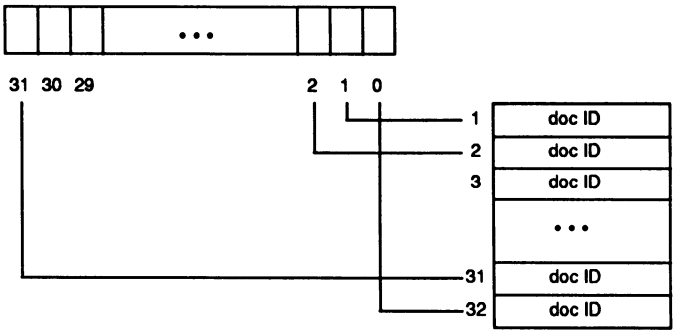
Since the entry for an extent is a fixed size structure, the entries for each edition are kept in a linked list. Editions are added to the list in chronological order, with the newest edition closest to the extent entry. Each edition in turn is a variable-size structure (the size of an extent's data can of course vary, and the number of formats written to the driver can vary for each edition) and is therefore also a linked list. The data format identifier for each chunk of data is stored in the block with the data as a 32-bit header.

The interest mask – is used to identify targets for a source extent. The position in the document-ID table serves as a bit number in this mask (i.e. slot 2 is bit 2 in the mask). In the extent's entry in the master table, the bits serve as identifiers of the target documents, and are only turned off by a target document severing a link. In the mask stored with each edition, the bits serve to record which documents have not yet read the data, and are turned off as soon as a target has read an edition. The read routine checks the interest mask for an edition after returning the data, and deletes the edition if no interested targets remain.

The document-ID table – is a table of unique document IDs. Every entry in this table is one of three types - Empty (zero), Target Document (1), or Source Document (a negative number). It is critical for every document interacting with the IAC driver to appear in this table, since the slot_ID is used to access the interest mask for extent entries; yet, document IDs are only assigned to source documents (a document that is only a target of others does not need to be identified by other than its slot_ID) - how to resolve this?

Target-only documents are those that do not pass a document ID to the "Complete Dependency" call; a dummy entry is put into the table to uniquely identify them for this session, and a slot_ID is assigned. Later, if they make an "Add Dependency" call and pass the existing slot_ID in (as they must under the protocol), the slot_ID is used to put the new document ID in the table, replacing the dummy entry. We take advantage of the fact that all date/times returned by the system are negative numbers to make this easy to process.

These data structures were designed for simplicity; they contain the minimum amount of information that will allow the driver to do its job. They do not, for example, maintain any



Slot IDs are non-zero and address thebits Modulo-32.
This is automatic when using the bit instructions of the 68000.

information about the "boundaries" of extents, since that is highly application specific and impossible to anticipate; also, maintaining that material in the driver would result in excessive traffic between the IAC driver and applications.

A Routine Commentary

Most of the stuff from last time that concentrated on the interface specifications was explained fairly well, and I don't wish to copy it here. Instead, I will provide my commentary on things that were perhaps not that clear. I'll go routine by routine.

Open – it turns out to be impossible to check for MultiFinder at boot time, when this driver is opened. Therefore, it is the responsibility of the application to check and refuse to try and support IAC operations when MultiFinder isn't running. The 'C' glue code provided does this check during the open call, so you shouldn't need to.

AddDependency —What's hard about this? Not much! Basically what it does is to add a new entry to the driver's internal tables that are used to keep track of data that is being made available to interested applications. This is called by the source application.

There are 2 ways in which a source application can make this call: with a default document ID of zero, or with existing document and slot IDs (the driver has no way of checking up on you, and assumes that non-zero IDs are valid). The default case is easy - just give the document it's IDs and send it home. If IDs are provided, however, life becomes complicated - there is the possibility of collisions with slot IDs already registered. If that occurs, a new slot ID is assigned and returned to the caller – this is why the slot ID returned from this call MUST ALWAYS be the one used in further communications with the driver.

A reminder that this call does NOT pass any data to the driver; an explicit "Write Data" call must be made.

CompleteDependency —This one is called by the "target" application, not the source! It tells the driver "hey, I'm interested in whatever's available right now." It basically lets the driver set up the "Interested mask" for the dependency properly. I'll say a bit more about the "Interested mask" later.

RemoveDependency —This one is a bit trickier, as it can be called by either the source application or the target. If it's called by the source, all editions of the data that have been posted get totally wiped, and the extent itself is deleted so that any targets

that were interested get informed that the data has been blasted out from under them (they'll get a "No such dependency" error). On the other hand, it may be a single target that's calling. If that is the case, the appropriate bit in the "Master Interest mask" (and all editions) gets reset (that's one less suitor, darling). When no one cares at all anymore (that is, the entire "Master Interest mask" goes to zero), the driver will blow away all of the editions just as it would if the source removed the dependency. Got all that? There's a quiz next month.

A reminder that this call does NOT retrieve any data from the driver; an explicit "Read Data" call must be made.

AvailableDependency —I had a truly bad pun for this one, but I'll resist the temptation. This is easy. It just makes the extent indicated by the parameters the source to all future *defaulted* "CompleteDependency"s until another dependency is made the available one. This has the highly desirable side-effect of making it trivial to have more than one target interested in the same source.

Status —Here's a simple one, too. It just tells whomever is calling how many documents have been registered with the driver, and how many extents that exist are relevant to the document asking for the status (have editions that the target hasn't read yet). Its real *raison d' etre*, though, is to tell the caller what version of the driver is installed (ahem).

Census —is like a high-powered Status. It doesn't care who's making the call. It just blasts out a list of all the extents that it knows of, period.

WriteData —this is rather like using the clipboard. The idea is that you may want to pass data in more than one format. A spreadsheet, for example, may wish to pass both the cell selection (let's call that type CELL) and a pie chart of the selection (as a PICT). Remember, this call is made by the source after it has established a dependency. The master interest mask is copied from the extent table to the header for the new edition to establish which targets have yet to read the new data.

ReadData — the target's analog of WriteData, of course. It's like SFGetFile inasmuch as it gives a list of preferred data types, in order of desirability. The target application makes this call, passing the most recent edition number of the data that it received last time. If there are one or more editions more recent than that, the driver will return the latest, marking the "Interested mask" for each successive edition as appropriate. It's conceivable (in fact, all too likely for some applications, such as databases) that the target won't be calling ReadData sufficiently frequently to get each and every edition that the source application posts, which is why we use editions in the first place. Reading an edition turns off the target's bit in the edition-interest-mask, and all older editions. An interest mask of zero results in that edition being deleted.

Notes on the code

This was developed on a Mac Plus running MPW 2.0.2 under System 6.0 It should work on any machine running MultiFinder, although the technique used in the glue to determine if MultiFinder is actually running is not authorized by Apple. Their claim that "you should only check for services you need" ignores the fact that in this case the service needed is the multitasking capability. A few other matters...

First, none of the people involved with this project belong to the "Names-cost-a-dollar-a-letter" school of identifiers. Hence, the verbose identifiers and extensive comments. It was hard enough to debug with almost every line commented, believe me!

Second, the INIT resource is set up so that if you have the object to ShowInit by Paul Mercer, you can modify a couple of lines and link it in to display start-up icons. We didn't have permission to print his source, and don't like publishing software in MacTutor without every line of source needed to rebuild it, so this seems a workable compromise.

Third, some of the makefiles may have traces of the Odesta development environment showing through, even though I tried to "genericize" them. I hope they won't be too hard to correct..

Bug Fixes and Updates

Inevitably, bugs will be found in this code, and we already have thought of several desirable improvements to add in the future (after we're recovered from this exercise in hubris). It is not reasonable to expect Dave Smith and MacTutor to act as the central exchange for this driver, since the effort involved could become significant.

After January 1st, 1989, we will be acting as IAC Central, issuing revised versions as necessary, etc. Bug reports can be sent to us immediately (we'd like to think there will never be any, but...) along with suggestions for improvements. We can be reached by electronic mail (please, no phones calls!!) at the following addresses:

Paul Snively – GEnie: PSNIVELY
(The first line of any letter should be:
ATTN: PSNIVELY PAUL SNIVELY for the mail server he is using)
Frank Alviani – GEnie: TOOLSMITH
Delphi: TOOLSMITH

In addition, US Snail can reach us (slowly) at the following address:

CodeSmiths
P.O. Box 8744
Waukegan, Ill 60079

We will try and respond in a timely fashion, given the constraints of very demanding jobs, a house renovation, and small children.

That about wraps it up for our end. Here —at long last!— is the source code for the driver and the glue (in 'C') to simplify writing applications to support the protocol!

Listing: driver.m

```
# Makefile for the IAC Driver!
# Frank Alviani  - 8/88

ob = {hlxEtc}  #Set these definitions to your normal working
directories
sr = {hlxSrc}
```

```
prog = {hlx}
e  = echo

"{prog}SAWSInit" ƒ  "{ob}SAWSdrvr.lnk" "{ob}SAWSInit"
"{sr}SAWSINIT.r"
  (e) 'Date -a -t' Rez IAC >> "{log}"
  Rez -t INIT -c SAWS -o "{prog}SAWSInit" "{sr}SAWSinit.r"

"{ob}SAWSDRVR.a.o" ƒ  "{sr}SAWSDRVR.a"
  (e) 'Date -a -t' Asm SAWSDRVR.a >> "{log}"
  Asm "{sr}SAWSDRVR.a" -i Me:MPW:AIncludes: -o "{hlxetc}"

"{ob}SAWSdrvr.lnk" ƒ  "{ob}SAWSDRVR.a.o"
  (e) 'Date -a -t' Link SAWSDRVR.a. >> "{log}"
  Link -t INIT -c SAWS -rt DRVR=31 -sg .IAC "{ob}SAWSDRVR.a.o"
∂
    -o "{ob}SAWSdrvr.lnk"

"{ob}SAWSINIT.a.o" ƒ  "{sr}SAWSINIT.a"
  (e) 'Date -a -t' Asm SAWSINIT.a >> "{log}"
  Asm "{sr}SAWSINIT.a" -i Me:MPW:AIncludes: -o "{hlxetc}"

"{ob}SAWSInit"  ƒ  "{ob}SAWSINIT.a.o"
  (e) 'Date -a -t' Link SAWSINIT.a >> "{log}"
  Link -rt INIT=0 -ra =16 "{ob}SAWSINIT.a.o" -o "{ob}SAWSInit"
```

Listing: SysEnv.a

```
;
; Glue for SysEnvirons that was skipped by Apple
; F. Alviani
; 7/88
;

    INCLUDE  'TRAPS.A'  ;Include Memory manager macros.
    PRINT ON

SYSENVIRONSFUNC EXPORT
    movea.l 4(sp),a0  ;a0 contains ptr to world record
    move.w 8(sp),d0 ;d0 contains desired version
    _SysEnvirons
    movea.l  (sp)+,a0 ;get return addr, remove
    lea 6(sp),sp ;pop parameters
    move d0,(sp)    ;put result in place
    jmp (a0)  ;go home

    DC.B 'SYSENVIR'

    END
```

Listing: IAC.h

```
/*This is the set of externs needed to access the IAC inter-
face routines */
/*The pointer/handle indicators are only reminders */

extern short  iac_add_dependency();        /* *doc_ID,
*slot_ID, *hat_check,
                                              *edition */
extern short  iac_available_dependency(); /* doc_id, hat_check
*/
extern short  iac_census();                /* *extent_count,
**extent_info */
extern short  iac_complete_dependency(); /* *doc_id, *slot_id,
*hat_check */
extern short  iac_open();
extern short  iac_read_data();             /* doc_id, slot_id,
hat_check, *edition,

 fmt_pref[], *fmt_code, **ext_data */
extern short  iac_remove_dependency();   /* doc_id, slot_id,
hat_check */
extern short  iac_status();                /* slot_id,
*vers_id, *doc_count,

 *extent_count */
extern short  iac_write_data();            /* doc_id,
hat_check, *edition,

 fmt_count, **ext_data */

/*error codes for iac_open, etc. */
# define NO_DRIVER -1
# define EARLY_SYS -2
# define MAX_EXTS 64

typedef struct {
    long   doc_ID;
    short    hat_check;
    short    ed_level;
} info_rec;

typedef struct {
    info_rec ext_entry[MAX_EXTS];
} info_tbl, *info_tblP, **info_tblH;
```

Listing: IAC.c

```
/***
 *
 * File:   IAC.c
 *
 * Package:  Inter Application Communications
 *
 * Description: interface package to the driver for the use of
 *       application programs.
 *
 * Author(s):
 *FEA    6/19/88
 *
 */

# include  <types.h>
# include  <files.h>
# include  <memory.h>
# include  <osutils.h>
# include  <serial.h>
# include  <iac.h>

# define MIN_BLK_SIZE 0xC
# define DEBUG  false

static short  iac_ref_num;

/**
 * Routine: iac_open
 *
 *  The IAC driver is opened by this call and the ioRefNum
 *  saved so it doesn't need to be passed with all calls.
 *  A null value is returned if the open is successful;
 *otherwise the operating system error is
 *  returned.
 */

short  iac_open()

(
SysEnvRec  theWorld;
/* ALMOST complete knowledge about the world */
THz      s_ZoneP, a_ZoneP; /* so we can check zone adjacency
*/
OSErr    the_err;

short    result = 0;

  the_err = SysEnvirons(1, &theWorld);
  if (theWorld.systemVersion < 0x0420)    /* too early! */
```

```
  {
    result = EARLY_SYS;    /* "no multifinder" */
  }
  else
  {
    /*check for multifinder active by checking if system zone
  is adjacent to application zone, which never happens under
      MultiFinder.
    */
    s_ZoneP = SystemZone();
    a_ZoneP = ApplicZone();
    if ( ((long) s_ZoneP->bkLim + MIN_BLK_SIZE) == (long)
a_ZoneP )
    {
      result = EARLY_SYS;
/* zones adjacent - no multifinder */
    }
    else /* now try to actually open the IAC driver */
    {
      SysBeep(1);
      result = OPENDRIVER("\p.IAC", &iac_ref_num);
    }
  }

  return(result);
}


/**
 * Routine: iac_add_dependency
 *
 *  This is used to inform the IAC driver that a new
 *dependency has been established and should be
 *added to its internal tables. A null value is
 *returned if the open is successful; otherwise the operating
 *system error is returned.
 */

short  iac_add_dependency(doc_id, slot_id, hat_check, edition)
  long *doc_id;
/* identifies the source document (permanent) */
  short  *slot_id;
/* identifies the source document (session) */
  short  *hat_check;/* extent identifier */
  short  *edition;
/* how many times extent has changed (session) */

{
IOParam    the_blk;
OSErr    the_err;

struct {
  short  func;
  long doc_id;
  short  slot_id;
  short  hat_check;
  short  edition;
} my_params;

# if DEBUG
  return(the_err=noErr);/* short circuit testing without MF */
# endif

  my_params.func = 1;      /* set up private parameter block
*/
  my_params.doc_id = *doc_id;
  my_params.slot_id = *slot_id;
  my_params.hat_check = *hat_check;

  the_blk.ioCompletion = nil;/* setup driver parametr block */
  the_blk.ioRefNum = iac_ref_num;
  the_blk.ioBuffer = &my_params;
  the_blk.ioReqCount = sizeof(my_params);
  the_blk.ioPosMode = fsFromStart;
  the_blk.ioPosOffset = 0;
  the_err = PBWrite(&the_blk.qLink, false);
/* add the dependency */

  if (the_err == noErr)   /* update output parameters */
  {
    *doc_id = my_params.doc_id;
    *slot_id = my_params.slot_id;
    *hat_check = my_params.hat_check;
    *edition = my_params.edition;
  }

  return(the_err);
}


/**
 * Routine: iac_complete_dependency
 *
 *  This is used to inform the IAC driver of a document that
 *  is interested in a particular dependency. A null value is
 *  returned if the open is successful; otherwise the
 *  operating system error is returned.
 */

short  iac_complete_dependency(doc_id, slot_id, hat_check)
  long *doc_id;
/* identifies the source document (permanent) */
  short  *slot_id;
/* identifies the source document (session) */
  short  *hat_check;/* extent identifier */

{
IOParam    the_blk;
OSErr    the_err;

struct {
  short  func;
  long doc_id;
  short  slot_id;
  short  hat_check;
} my_params;

# if DEBUG
  return(the_err=noErr);
/* short circuit for testing without MF */
# endif

  my_params.func = 2;      /* set up private parameter block
*/
  my_params.doc_id = *doc_id;
  my_params.slot_id = *slot_id;
  my_params.hat_check = *hat_check;

  the_blk.ioCompletion = nil;/* setup driver parametr block */
  the_blk.ioRefNum = iac_ref_num;
  the_blk.ioBuffer = &my_params;
  the_blk.ioReqCount = sizeof(my_params);
  the_blk.ioPosMode = fsFromStart;
  the_blk.ioPosOffset = 0;
  the_err = PBWrite(&the_blk.qLink, false);
/* complete the dependency */

  if (the_err == noErr)   /* update output parameters */
  {
    *doc_id = my_params.doc_id;
    *slot_id = my_params.slot_id;
    *hat_check = my_params.hat_check;
  }

  return(the_err);
}
```

```
/**
 * Routine: iac_remove_dependency
 *
 *   This is used to inform the IAC driver that a document is
 *   no longer interested in a particular dependency. If the
 *   document is the original source all "targets" will be
 *   informed that there is no longer any such
 *extent; if all targets lose interest the source will be
 * informed that it no longer needs to update the extent.
 */

short  iac_remove_dependency(doc_id, slot_id, hat_check)
  long doc_id;
/* identifies the source document (permanent) */
  short  slot_id;
/* identifies the source document (session) */
  short  hat_check; /* extent identifier */

{
IOParam    the_blk;
OSErr    the_err;

struct {
  short  func;
  long doc_id;
  short  slot_id;
  short  hat_check;
} my_params;

# if DEBUG
  return(the_err=noErr);/* short circuit testing without MF */
# endif

  my_params.func = 3; /* set up private parameter block */
  my_params.doc_id = doc_id;
  my_params.slot_id = slot_id;
  my_params.hat_check = hat_check;

  the_blk.ioCompletion = nil; /* setup driver paramtr block */
  the_blk.ioRefNum = iac_ref_num;
  the_blk.ioBuffer = &my_params;
  the_blk.ioReqCount = sizeof(my_params);
  the_blk.ioPosMode = fsFromStart;
  the_blk.ioPosOffset = 0;
  the_err = PBWrite(&the_blk.qLink, false);
/* remove the dependency */

  return(the_err);
}


/**
 * Routine: iac_available_dependency
 *
 *   This sets the indicated extent as the "available extent"
 *to be used as the source for future defaulted
 *"complete_dependency"  calls.
 */

short  iac_available_dependency(doc_id, hat_check)
  long doc_id;/* identifies the source document (permanent) */
  short  hat_check; /* extent identifier */

{
IOParam    the_blk;
OSErr    the_err;

struct {
  short  func;
  long doc_id;
  short  hat_check;
} my_params;

# if DEBUG
  return(the_err=noErr);
/* short circuit for testing without MF */
# endif

  my_params.func = 4;      /* set up private parameter block
*/
  my_params.doc_id = doc_id;
  my_params.hat_check = hat_check;

  the_blk.ioCompletion = nil;/* setup driver parametr block */
  the_blk.ioRefNum = iac_ref_num;
  the_blk.ioBuffer = &my_params;
  the_blk.ioReqCount = sizeof(my_params);
  the_blk.ioPosMode = fsFromStart;
  the_blk.ioPosOffset = 0;
  the_err = PBWrite(&the_blk.qLink, false);
/* this dependency is now "available" */

  return(the_err);
}


/**
 * Routine: iac_status
 *
 *   This allows the application program to find out what's
 *going on.
 */

short  iac_status(slot_id, vers_id, doc_count, extent_count)
  short  slot_id;   /* identifies the inquiring document
(session) */
  short  *vers_id;     /* driver version*100 */
  short  *doc_count;  /* count of active documents */
  short  *extent_count;  /* count of extents relevant to
inquiring doc */

{
IOParam    the_blk;
OSErr    the_err;

struct {
  short  func;
  short  slot_id;
  short  vers_id;
  short  doc_count;
  short  extent_count;
} my_params;

# if DEBUG
  return(the_err=noErr);/* short circuit testing without MF */
# endif

  my_params.func = 5;      /* set up private parameter block
*/
  my_params.slot_id = slot_id;

  the_blk.ioCompletion = nil;/* setup driver parametr block */
  the_blk.ioRefNum = iac_ref_num;
  the_blk.ioBuffer = &my_params;
  the_blk.ioReqCount = sizeof(my_params);
  the_blk.ioPosMode = fsFromStart;
  the_blk.ioPosOffset = 0;
  the_err = PBRead(&the_blk.qLink, false);
/* read IAC status */

  if (the_err == noErr)   /* update output parameters */
  {
    *vers_id = my_params.vers_id;
    *doc_count = my_params.doc_count;
    *extent_count = my_params.extent_count;
  }

  return(the_err);
```

```
}


/**
 * Routine: iac_census
 *
 *  This provides identifying info for all registered extents.
 */

short  iac_census(extent_count, extent_info)
  short    *extent_count;  /* count of extents registered */
  info_tblP  extent_info;  /* Ptr to table of info for each
extent */

{
IOParam    the_blk;
OSErr    the_err;
short    i;

struct {
  short    func;
  short    extent_count;
  info_rec extent_info[MAX_EXTS];
} my_params;

# if DEBUG
  return(the_err=noErr);  /* short circuit for testing
without MF */
# endif

  my_params.func = 6;      /* set up private parameter block
*/

  the_blk.ioCompletion = nil;       /* set up driver parameter
block */
  the_blk.ioRefNum = iac_ref_num;
  the_blk.ioBuffer = &my_params;
  the_blk.ioReqCount = sizeof(my_params);
  the_blk.ioPosMode = fsFromStart;
  the_blk.ioPosOffset = 0;
  the_err = PBRead(&the_blk.qLink, false);
/* read IAC status */

  if (the_err == noErr) /* update output parameters */
  {
    *extent_count = my_params.extent_count;
    BlockMove (&my_params.extent_info[0],
           (Ptr) extent_info,
           (long) my_params.extent_count * sizeof(info_rec));
  }

  return(the_err);
}


/**
 * Routine: iac_write_data
 *
 *  This updates the data for the specified extent, resulting
 *in a new change level.
 */

short  iac_write_data(doc_id, hat_check, edition, fmt_count,
ext_data)
  long doc_id;  /* identifies source document (permanent) */
  short  hat_check; /* extent identifier */
  short  *edition;  /* how many times extent has changed
(session) */
  short  fmt_count; /* number of formats being written */
  Handle ext_data;  /* Handle to actual data */

{
IOParam    the_blk;
OSErr    the_err;

struct {
  short  func;
  long doc_id;
  short  hat_check;
  short  edition;
  short  fmt_count;
  Handle the_dataH;
} my_params;

# if DEBUG
  return(the_err=noErr);  /* short circuit for testing
without MF */
# endif

  my_params.func = 7;      /* set up private parameter block
*/
  my_params.doc_id = doc_id;
  my_params.hat_check = hat_check;
  my_params.fmt_count = fmt_count;
  my_params.the_dataH = ext_data;

  the_blk.ioCompletion = nil;       /* set up driver parameter
block */
  the_blk.ioRefNum = iac_ref_num;
  the_blk.ioBuffer = &my_params;
  the_blk.ioReqCount = sizeof(my_params);;
  the_blk.ioPosMode = fsFromStart;
  the_blk.ioPosOffset = 0;
  the_err = PBWrite(&the_blk.qLink, false);
/* update dependency */

  if (the_err == noErr)   /* update output parameters */
  {
    *edition = my_params.edition;
  }

  return(the_err);
}


/**
 * Routine: iac_read_data
 *
 *  This is used to retrieve the actual data for the latest
 *change_level for the specified extent. The IAC driver will
 *record that the inquiring document has read the data.
 *
 *ext_data will be resized by the driver to hold the data.
 */

short  iac_read_data(doc_id, slot_id, hat_check, edition,
fmt_pref,
          fmt_code, ext_data)
  long doc_id;      /* identifies the source document (perma-
nent) */
  short  slot_id;   /* identifies the source document (ses-
sion) */
  short  hat_check;    /* extent identifier */
  short  *edition;     /* how many times extent has changed
(session) */
  long fmt_pref[3]; /* preferred formats, descending desira-
bility */
  long *fmt_code;   /* format returned to caller */
  Handle ext_data;     /* Handle to actual data */

{
IOParam    the_blk;
OSErr    the_err;

struct {
  short  func;
  long doc_id;
  short  slot_id;
```

```
  short  hat_check;
  short  edition;
  long fmt_pref[3];
  long fmt_code;
  Handle ext_data;
} my_params;

# if DEBUG
  return(the_err=noErr);  /* short circuit for testing
without MF */
# endif

  my_params.func = 8;     /* set up private parameter block
*/
  my_params.doc_id = doc_id;
  my_params.slot_id = slot_id;
  my_params.hat_check = hat_check;
  my_params.edition = *edition;
  my_params.fmt_pref[0] = fmt_pref[0];
  my_params.fmt_pref[1] = fmt_pref[1];
  my_params.fmt_pref[2] = fmt_pref[2];
  my_params.ext_data = ext_data;

  the_blk.ioCompletion = nil;      /* set up driver parameter
block */
  the_blk.ioRefNum = iac_ref_num;
  the_blk.ioBuffer = &my_params;
  the_blk.ioReqCount = sizeof(my_params);
  the_blk.ioPosMode = fsFromStart;
  the_blk.ioPosOffset = 0;
  the_err = PBRead(&the_blk.qLink, false);/* read the data */

  if (the_err == noErr)
  {
    *edition = my_params.edition;
    *fmt_code = my_params.fmt_code;
  }

  return(the_err);
}
```

Listing: SAWSDRVR.a

```
;*****************************************************************
;***** The SAWS Inter-Application Communication Driver
*****
;***** Written with blazing speed 6-8/88 by Paul F. Snively
;***** With one HECK of a lotta help from Frank Alviani
***
;*****          and Mike Walsh                              *****
;*****************************************************************
;
;Modification History:
;6/19/88 First Draft-1.0D1-Paul F. Snively
;7/10/88 Second draft-1.0B1-Paul F. Snively & Michael R.
;Walsh
;8/7/88  Yet another crack at it-1.0B1-Paul F. Snively &
; Frank Alviani
;8/8-21/88 Final Debugging - Frank Alviani
;
;Basically this puppy is the DRVR resource that actually
;implements Frank's cool ideas that are documented in the
; article that accompanies this listing.
;

    INCLUDE  'Traps.a'
    INCLUDE  'QuickEqu.a'
    INCLUDE  'SysEqu.a'
    INCLUDE  'SysErr.a'
    INCLUDE  'ToolEqu.a'
    STRING PASCAL

NumOfDocs  EQU 32   ;Max of 32 documents
NumOfExtents EQU 64

Unimpl EQU $9F    ;For testing environment
OSDisp EQU $8F
DummyID  EQU 1

NoMoreDocs EQU -2000    ;This one isn't used-yet
NoMoreSlotsEQU -2001    ;Neither is this one-yet
WriteFailedEQU -2002    ;Or this one...
MissingLinkEQU -2003;Yet another SAWS-defined error
NoNewerEdition EQU -2004    ;YASAWSE
ReadFailed EQU -2005    ;SASAWSE
NoSuchDep  EQU -2006
OldROMs  EQU -2007

ExtentRec  RECORD 0
DocID  DS.L 1;Home document for this extent
HatCheck DS.W 1   ;Unique ID for this extent
Edition  DS.W 1        ;The edition number
MstrInterestMsk DS.L 1        ;Master Interest Mask
EditionListDS.L 1        ;Edition List
ExtentSize EQU *
    ENDR

EditionHdr RECORD 0
NextEd DS.L 1 ;Handle to next Ed
InterestMask  DS.L 1 ;Mask for this particular Ed
NumFormats DS.W 1 ;How many formats stored?
EdInstancesDS.L 0 ;Instances start here
EditionSizeEQU *
    ENDR

IACGlobals RECORD 0
ExtentCountDS.W 1 ;Number of extents extant
ExtentTableDS.L NumOfExtents*ExtentRec.ExtentSize
DocIDs DS.L NumOfDocs;Unique Document identifiers
AvailDependency DS.L 1 ;Offset of currently available depend-
ency
IACGlobalsSize  EQU *
    ENDR

DriverEntryPROC     ; See Device Manager IM:2
    IMPORT DriverOpen,DriverPrime
    IMPORT DriverCtl,DriverDone
    IMPORT DriverClose

  DC.B (1<<dReadEnable) + (1<<dWritEnable); We handle reads/
writes
  DC.B 0    ; Lower byte is unused
  DC.W 0*60   ; 0 sec periodic update
  DC.W 0    ; We don't do events
  DC.W 0    ; No menu for this accessory

  DC.W DriverOpen-DriverEntry     ; Open routine
  DC.W DriverPrime-DriverEntry      ; Prime
  DC.W DriverCtl-DriverEntry      ; Control
  DC.W DriverDone-DriverEntry     ; Status - unused
  DC.W DriverClose-DriverEntry      ; Close

  DC.B '.IAC'       ;The name of our driver
  ALIGN 2       ; Word align

    ENDPROC

DriverOpen PROC
    IMPORT IACGlobalsSize

  TST.W  ROM85      ;old ROMs?
  BGE.S  @1         ;no, keep going?
  MOVE.W #OldROMs,D0
  BRA.S  ExitOpen
@1
  with   IACGlobals
  TST.L  dCtlStorage(A1)     ;Did we already allocate our
                space?
  BNE.S  ExitOpen   ;If so, then we're already open
```

```
  MOVE.L #IACGLobalsSize,D0  ;How much global space do we
              want?
  _NewHandle sys,clear  ;Try to allocate it
  BNE.S  ExitOpen ;Fail miserably since error
          occurred
@2
  MOVE.L A0,dCtlStorage(A1)  ;Otherwise store handle
  MOVE.W #0,ExtentCount    ;We start with zero extents
  endwith    ;IACGlobals

ExitOpen
  RTS        ;We're outta here
    ENDPROC

DriverClosePROC
    IMPORT WipeExtentData
  MOVE.L dCtlStorage(A1),D0  ;Get our global handle
  BEQ.S  ExitClose     ;If we don't have one, leave!
  MOVEA.L  D0,A6       ;Otherwise get into address
            register

  with   IACGlobals
  MOVE.W ExtentCount(A6),D7  ;How many extents are there?
  LEA  ExtentTable(A6),A0  ;Point to the ExtentTable
  MOVE.L A0,D6       ;Get it in the right register
  endwith    ;IACGlobals

WalkExtents
  MOVEA.L  D6,A0       ;Get Extent Pointer
  JSR  WipeExtentData    ;Kill its data

NextExtent
  with   ExtentRec
  SUBQ.W #1,D7       ;Decrement extent counter
  BEQ.S  EndClose    ;And continue until done
  ADD.L  #ExtentSize,D6    ;Point to next extent
  BRA.S  WalkExtents  ;And loop
  endwith    ;ExtentRec

EndClose
  MOVEA.L  dCtlStorage(A1),A0  ;Get our globals handle
  _DisposHandle      ;And get rid of it

ExitClose
  RTS        ;Back to caller
    ENDPROC

;
; This is the main entry point for ALL functions of the driver
;
DriverPrimePROC
    IMPORT DriverAddDependency,DriverCompleteDependency
    IMPORT DriverRemoveDependency,DriverRemoveDependency
    IMPORT DriverAvailableDependency,DriverStatus
    IMPORT DriverCensus,DriverWriteData
    IMPORT DriverReadData

  MOVEA.L  dCtlStorage(A1),A6  ;Get our global handle
  MOVEA.L  ioBuffer(A0),A4   ;Get the address of the "data"
  MOVE.W (A4),D0       ;Get the routine identifier
  ADD.W  D0,D0       ;Multiply it by two
  MOVE.W RoutineTable(D0),D0 ;Get routine offset
  JSR  RoutineTable(D0)  ;Go to the proper routine
  MOVE.L JIODone,-(A7)  ;wrap up
  RTS
RoutineTable
  DC.W 0       ;1-based function codes...
  DC.W DriverAddDependency-RoutineTable
  DC.W DriverCompleteDependency-RoutineTable
  DC.W DriverRemoveDependency-RoutineTable
  DC.W DriverAvailableDependency-RoutineTable
  DC.W DriverStatus-RoutineTable
  DC.W DriverCensus-RoutineTable
  DC.W DriverWriteData-RoutineTable
  DC.W DriverReadData-RoutineTable

  ENDPROC

;
; This routine adds the source of a link to the dependency
table
;
DriverAddDependency PROC
      IMPORT FindSlotID

AddDepRec  RECORD 0
Func   DS.W 1
docID    DS.L 1
slot_ID    DS.W 1
hatCheck DS.W 1
edition    DS.W 1
AddDepRecSize EQU *
    ENDR

  MOVEA.L  (A6),A2         ;Get globals pointer
  LEA  IACGlobals.DocIDs(A2),A3 ;Point past end of
                    extentTable
  LEA  IACGlobals.ExtentTable(A2),A2;Point to base of
                    extent table
  MOVEQ  #0,D2         ;Clear our counter
AddDepLoop
  TST.L  ExtentRec.docID(A2)   ;Is this slot available?
  BEQ.S  FoundAvailSlot      ;If so, we're done looking
  ADD.L  #ExtentRec.ExtentSize,D2 ;Maintain "available"
                    offset
  ADDA.L #ExtentRec.ExtentSize,A2 ;Else point to next
                    record
  CMPA.L A2,A3         ;Are we done?
  BGT.S  AddDepLoop      ;If not, look again

AddDepFail
  MOVE.W #NoMoreDocs,D0      ;Return custom OSErr
  BRA  AddDepExit      ;And leave

FoundAvailSlot
  MOVE.L AddDepRec.docID(A4),D0  ;Get the docID
  BNE.S  ExistingDocID     ;Go if it already exists
  MOVE.L Time,D0         ;Else copy it from low RAM
  MOVE.L D0,AddDepRec.docID(A4)  ;Update param block

ExistingDocID
  MOVE.L D0,ExtentRec.docID(A2)  ;Store the docID
  MOVE.L D0,D5;        ;Save for later
  MOVE.L A2,D4         ;Save extent-rec addr for
                    later
  MOVE.W AddDepRec.slot_ID(A4),D1;Did we get passed a
                    slot_ID?
  BEQ.S  NeedASlot       ;Go get one if not
  MOVE.L (A6),A3
  LEA  IACGlobals.DocIDS(A3),A3 ;Pt to table of unique
                    IDs
  SUBQ.W #1,D1         ;Adjust index to 0-based
  LSL.W  #2,D1         ;Convert to offset
  CMPI.L #DummyID,0(A3,D1.W)   ;Is passed slot_ID target
                    only?
  BEQ.S  HaveASlot       ;Yes, put in real docID
  CMP.L  0(A3,D1.W),D5     ;ID in table OK?
  BEQ.S  HaveASlot       ;Yes, use slotID as is
                      NeedASlot
  BSR FindSlotID       ;Get a valid slot ID
  BLT.S  AddDepExit      ;No slots - Leave
                    HaveASlot
  MOVE.L D5,0(A3,D1.W)     ;Save docID in slot
  MOVEA.L  (A6),A2         ;Dereference global handle
  MOVE.W D0,AddDepRec.slot_ID(A4);Store the slot_ID
  MOVE.W AddDepRec.hatCheck(A4),D1  ;Is there already a
                    hatCheck?
  BNE.S  StuffHatCheckCalc   ;If so, bypass calculation
```

```
  LEA IACGlobals.docIDs(A2),A3 ;Point past the end
  LEA IACGlobals.ExtentTable(A2),A2;Point to right
                        place

; Search for highest hatCheck registered for this docID
  MOVEQ #0,D1          ;Initialize counter
  MOVE.L AddDepRec.docID(A4),D0  ;Get the docID again

HatCheckLoop
  CMP.L ExtentRec.docID(A2),D0  ;Is the docID the same?
  BNE.S NotSame         ;No, ignore
  CMP.W ExtentRec.hatCheck(A2),D1 ;Compare hatChecks
  BGE.S NotSame         ;Go if no need for update
  MOVE.W ExtentRec.hatCheck(A2),D1 ;Save max hatCheck
                        NotSame
  ADDA.L #ExtentRec.ExtentSize,A2 ;Point to next record
  CMPA.L A2,A3          ;Are we done?
  BGT.S HatCheckLoop      ;Go if not
  ADDQ.W #1,D1          ;Bump to next hatCheck
  MOVE.W D1,AddDepRec.hatCheck(A4) ;Update param block

StuffHatCheckCalc
  MOVEA.L D4,A2         ;Point to "working" extent
  MOVE.W D1,ExtentRec.hatCheck(A2) ;Store hatCheck
  MOVE.W #0,AddDepRec.edition(A4);Pass back edition zero
  MOVE.W #0,ExtentRec.edition(A2);Set edition zero
  MOVEA.L (A6),A2         ;Get globals pointer
  MOVE.L D2,IACGlobals.AvailDependency(A2);It's also the
                        available dependency
  ADD.W #1,IACGlobals.ExtentCount(A2)
  MOVE.W #noErr,D0      ;We succeeded!
AddDepExit
  RTS

  ENDPROC

;
; This routine adds the destination to a link that's been
; started.  If a passed-in slot_ID conflicts with an existing
; entry, it is re-assigned
DriverCompleteDependencyPROC
    IMPORT FindSlotID

CompDepRec RECORD 0
Func  DS.W 1
docID   DS.L 1
slot_ID   DS.W 1
hatCheck DS.W 1
CompDepRecSize EQU *
    ENDR

  MOVEA.L (A6),A2         ;Dereference globals handle
  MOVE.L CompDepRec.docID(A4),D0   ;Get the passed docID
  BNE.S HaveDocID       ;Go if we were passed docID
  MOVE.L IACGlobals.AvailDependency(A2),D1;Get available
                          dependency
  LEA IACGlobals.ExtentTable(A2),A2;Point to extent
                        records
  MOVE.L ExtentRec.docID(A2,D1.L),D0 ;Get the docID
                        with CompDepRec
  MOVE.L D0,docID(A4)     ;Return to sender
HaveDocID
  MOVE.W slot_ID(A4),D1     ;Did we get passed a slot_ID?
  BNE.S ExistingSlotID      ;Go get one if not
  MOVE.L #DummyID,D3    ;Use dummy ID in slot for now
  BRA.S NeedSlotID      ;Go find slot to put it in
ExistingSlotID
  MOVE.L (A6),A3
  LEA IACGlobals.DocIDS(A3),A3 ;Pt to table of unique
                          IDs
  SUBQ.W #1,D1          ;Adjust index to 0-based
  LSL.W #2,D1           ;Convert to offset
  CMPI.L #DummyID,0(A3,D1.W)  ;Is passed slot_ID a dummy?
  BNE.S CheckSlotID     ;No, see if it's OK
  MOVE.L D0,0(A3,D1.W)    ;Yes, replace with real thing
CheckSlotID
  CMP.L 0(A3,D1.W),D0     ;DocIDs match?
  BEQ.S HaveSlotID
NeedSlotID
  BSR FindSlotID      ;Get a valid slot ID
  BLT.S CompDepExit     ;failed...
HaveSlotID
  MOVE.W D0,slot_ID(A4)      ;Store slot_ID in caller's
                        block
  MOVE.L docID(A4),D3
  endwith     ;CompDepRec
  MOVE.L D3,(A3,D1.W)      ;Store docID
  MOVEA.L (A6),A2         ;Dereference globals handle
  MOVE.L IACGlobals.AvailDependency(A2),D2;Get available
                          dependency index
  LEA IACGlobals.ExtentTable(A2),A2;Point to extent
                        records
  with  ExtentRec
  MOVE.L MstrInterestMsk(A2,D2.L),D3  ;Use register to
                        address all 32 bits
  BSET D0,D3          ;Set the interested bit
  MOVE.L D3,MstrInterestMsk(A2,D2.L)  ;Back into table
  MOVE.L EditionList(A2,D2.L),D1    ;Get the "Edition List
                        handle"
  BGT.S WalkEditions0   ;If it's a handle, deal with it
  MOVE.W #NoSuchDep,D0     ;set error code
  BEQ.S CompDepExit     ;If it's zero, we do nothing
  MOVE.L #0,EditionList(A2,D2.L)    ;Otherwise give it no
                        respect
  BRA.S CompDepExit
  endwith     ;ExtentRec

WalkEditions0
  MOVE.W ExtentRec.HatCheck(A2,D2.L),D3
  MOVE.W D3,CompDepRec.hatCheck(A4) ;Return to caller
WalkEditions
  MOVEA.L D1,A3         ;Copy the handle
  MOVEA.L (A3),A3         ;Dereference the handle
  with  EditionHdr
  MOVE.L InterestMask(A3),D3   ;So we can use 32-bit mode
  BSET D0,D3          ;Set the appropriate bit
  MOVE.L D3,InterestMask(A3)
  MOVE.L NextEd(A3),D1     ;Get handle to next handle
  endwith     ;EditionHdr
  BNE.S WalkEditions    ;If there is one, deal with it
  MOVE.W #noErr,D0      ;We succeeded!

CompDepExit
  RTS

  ENDPROC

;
; This routine severs a link and removes it from the tables
;
DriverRemoveDependencyPROC
    IMPORT FindExtentByDandHC,WipeExtentData

RemovDepRecRECORD 0
Func  DS.W 1
docID   DS.L 1
slotID  DS.W 1
hatCheck DS.W 1
RemovDepRecSize EQU *
    ENDR

  MOVEA.L (A6),A2         ;Dereference globals handle
  MOVE.L RemovDepRec.docID(A4),D0 ;Get the passed docID
  MOVE.W RemovDepRec.hatCheck(A4),D1 ;Get the passed
                          hatCheck
  JSR FindExtentByDandHC    ;Find the extent
  BNE.S RemovDepExit ;Use FEBAHDC's errcode and scram.
```

```
  MOVEA.L  (A6),A2          ;Dereference globals handle
  LEA  IACGlobals.ExtentTable(A2),A2;Get table base
  MOVE.L ExtentRec.MstrInterestMsk(A2,D2.L),D3        ;Get
                                interest mask
  MOVE.W RemovDepRec.slotID(A4),D4  ;Get the passed slotID
  BCLR D4,D3  ;Declare this slot uninterested
  BNE.S  RemovTarget  ;If bit WAS set, target was
          doing the removing…
RemovSource          ;…Otherwise it's the source.
  MOVEA.L  ExtentRec.EditionList(A2,D2.L),A0;Stuff list
                           header into A0
  JSR  WipeExtentData       ;Kill the extent's data
  LEA  IACGlobals.ExtentTable(A2),A2;Get table base
  with   ExtentRec
  MOVE.L #0,DocID(A2,D2.L)   ;Clear DocID
  MOVE.L #0,HatCheck(A2,D2.L)  ;Clear HatCheck/Edition
  MOVE.L #0,MstrInterestMsk(A2,D2.L)  ;Clear
                               MstrInterestMsk
  MOVE.L #0,EditionList(A2,D2.L)    ;Clear EditionList
  endwith     ;ExtentRec
  MOVE.W #noErr,D0       ;Set return code
  BRA.S  RemovDepCleanup
RemovTarget
  TST.L  D3       ;Does anybody care?
  BEQ.S  RemovSource      ;If no, then go to sleep
  MOVE.W #noErr,D0  ;Set return code
RemovDepCleanup
  MOVE.W RemovDepRec.slotID(A4),D4  ;Get the passed slotID
  SUBQ.W #1,D4
  LSL.W  #2,D4         ;Turn into offset
  MOVEA.L  (A6),A2          ;Dereference globals handle
  with   IACGlobals
  SUBQ.W #1,ExtentCount(A2)  ;One less extent in world
  LEA  DocIDs(A2),A2      ;Point to table of unique IDs
  MOVE.L #0,0(A2,D4.W)      ;Clear doc's slot
  endwith     ;IACGlobals
RemovDepExit
  RTS           ;Bye-bye

  ENDPROC

;
; This routine marks a link source as "available for
; completion"
;
DriverAvailableDependency  PROC
     IMPORT FindExtentByDandHC

AvailDepRecRECORD 0
Func   DS.W 1
docID     DS.L 1
hatCheck DS.W 1
AvailDepRecSize EQU  *
     ENDR

  MOVEA.L  (A6),A2          ;Dereference globals handle
  MOVE.L AvailDepRec.docID(A4),D0;Get the passed docID
  MOVE.W AvailDepRec.hatCheck(A4),D1  ;Get the passed
                           hatCheck
  JSR  FindExtentByDandHC     ;Get dependancy if it
                         exists
  BNE.S  AvailDepExit       ;If subr plotzed get out.
  MOVEA.L  (A6),A2
  MOVE.L D2,IACGlobals.AvailDependency(A2);Otherwise, we
                           have a winner!
  MOVE.W #noErr,D0       ;Set return code
AvailDepExit
  RTS           ;Bye-bye

  ENDPROC

;
; This routine lets the client know what's happening
;
DriverStatus PROC

StatusRec  RECORD 0
Func   DS.W 1
slotID   DS.W 1
versID   DS.W 1
docCount DS.W 1
extentCountDS.W 1
StatusRecSizeEQU  *
     ENDR

  MOVEA.L  (A6),A2          ;Dereference globals handle
  MOVE.W StatusRec.slotID(A4),D0    ;Get slotID
  MOVE.W #100,StatusRec.versID(A4)  ;Report version as '100'
  LEA  IACGlobals.AvailDependency(A2),A3;Point past end
  LEA  IACGlobals.DocIDs(A2),A2;Point to extent table
  MOVEQ  #0,D3         ;Clear our counter
CountDoxLoop
  TST.L  0(A2)         ;Is the docID for real?
  BGE.S  CountDoxSkip      ;If neg, then a modern date
  ADDQ #1,D3          ;…so add 1 to the count
CountDoxSkip
  ADDA.L #4,A2         ;Else point to next record
  CMPA.L A2,A3         ;Are we done?
  BGT.S  CountDoxLoop      ;If not, look again
  MOVE.W D3,StatusRec.docCount(A4)  ;Report number of docs

  MOVEA.L  (A6),A2          ;Dereference globals handle
  LEA  IACGlobals.DocIDs(A2),A3;Point past end
  LEA  IACGlobals.ExtentTable(A2),A2;Point to extent
                           table
  MOVEQ  #0,D3         ;Clear our counter
CountExtLoop
  with   ExtentRec
  MOVE.L MstrInterestMsk(A2),D4  ;Fetch interest mask
  BTST.L D0,D4         ;Is this slot interested?
  BEQ.S  CountExtSkip      ;If not, then get next extent
  MOVEA.L  EditionList(A2),A0     ;Get EditionList HANDLE
  endwith     ;ExtentRec
FindExtLoop
  MOVE.L A0,D4         ;valid address?
  BEQ.S  CountExtSkip      ;No more editions, get out
  MOVEA.L  (A0),A0
  MOVE.L EditionHdr.InterestMask(A0),D4
  BTST.L D0,D4  ;Is this edition interested?
  BEQ.S  FindExtSkip     ;No, so check next extent
  ADDI.L #1,D3         ;Yes, add 1 and…
  BRA.S  CountExtSkip      ;…Get out
FindExtSkip
  MOVEA.L  EditionHdr.NextEd(A0),A0 ;Move next EditionList
                        in
  BRA.S  FindExtLoop     ;And test back there…
CountExtSkip
  ADDA.L #ExtentRec.ExtentSize,A2;Else point to next
                        record
  CMPA.L A2,A3         ;Are we done?
  BGT.S  CountExtLoop      ;If not, look again
  MOVE.W D3,StatusRec.extentCount(A4) ;Report number of
                           extents
StatusExit
  MOVE.W #0,D0         ;Indicate exit OK
  RTS

  ENDPROC

;
; This routine returns a count of active programs, links, etc.
to the client
;
DriverCensus PROC

Census1Rec RECORD 0
docID     DS.L 1
edition     DS.W 1
```


© The Definitive MacTutor, Vol. 4

```
hatCheck DS.W 1
Census1RecSize  EQU *
    ENDR

CensusRec  RECORD Ø
Func   DS.W 1
extentCountDS.W 1
Census1    DS.B Census1Rec
CensusRecSizeEQU *
    ENDR

  MOVEA.L  (A6),A2        ;Dereference globals handle
  MOVE.L #CensusRec.Census1,D4   ;Offset to write for
                output recs
  LEA  IACGlobals.DocIDs(A2),A3 ;Point past end
  LEA  IACGlobals.ExtentTable(A2),A2;Point to extent
                table
  MOVEQ  #Ø,D3        ;Clear our counter
CensusLoop
  TST.L  ExtentRec.docID(A2)   ;Check this extent's docID
  BEQ.S  CensusSkip     ;If wrong, skip to next
  with   ExtentRec
  MOVE.L docID(A2),Census1Rec.docID(A4,D4.L) ;Move docID
  MOVE.L Edition(A2),Census1Rec.edition(A4,D4.L)     ;Move
                edition
  MOVE.L HatCheck(A2),Census1Rec.hatCheck(A4,D4.L)   ;Move
                hatCheck
  endwith    ;ExtentRec
  ADDI.L #Census1Rec.Census1RecSize,D4;Move to next output
                rec
  ADDQ.W #1,D3
CensusSkip
  ADDA.L #ExtentRec.ExtentSize,A2 ;Else point to next
                record
  CMPA.L A2,A3        ;Are we done?
  BGT.S  CensusLoop     ;If not, look again
CensusExit
  MOVE.W D3,CensusRec.extentCount(A4) ;Report extent count
  MOVE.W #noErr,DØ      ;Nothing is wrong
  RTS          ;Bye-bye

  ENDPROC

;
; This routine is called by the client to write data to the
; driver
DriverWriteData PROC
    IMPORT FindExtentByDandHC

WriteRec RECORD Ø
Func   DS.W 1
docID    DS.L 1
hatCheck DS.W 1
edition    DS.W 1
formatCountDS.W 1
theData    DS.L 1
WriteRecSize  EQU *
    ENDR

FmtDataBlock  RECORD Ø
formatCode DS.L 1
dataSize DS.L 1
myData   DS.L 1
    ENDR

  MOVEA.L  (A6),A2        ;Dereference globals handle
  MOVE.L WriteRec.docID(A4),DØ   ;Get the passed docID
  MOVE.W WriteRec.hatCheck(A4),D1;Get the passed hatCheck
  JSR  FindExtentByDandHC    ;Get dependancy if it
                exists
  BEQ.S  TableSetup     ;Get going. Otherwise,
  RTS          ;subr plotzed…leave w/o popping A1

TableSetup
  MOVEA.L  (A6),A2        ;Dereference globals handle
  LEA  IACGlobals.ExtentTable(A2),A2;Get table base
  MOVE.W WriteRec.formatCount(A4),D5  ;Get the passed
                formatCount
  EXT.L  D5
  ASL.L  #2,D5        ;Make it an offset
  MOVE.L A1,-(SP)     ;preserve dCtlStorage Ptr
  MOVEA.L  WriteRec.theData(A4),A1    ;Get the passed
                theData
  MOVEA.L  A1,AØ
  _HLock          ;Freeze the user data
  MOVEA.L  (A1),A1        ;…And put the pointer in A1

  MOVE.L #EditionHdr.EditionSize,DØ ;How much global space
                do we want?
  ADD.L  D5,DØ        ;Add format entries
  _NewHandle sys,clear;Try to allocate it (sys
            temporary)
  BNE  WriteDataFailed     ;…Couldn't
  MOVE.L AØ,D7        ;preserve

  MOVEA.L  (A6),A2        ;Dereference globals handle
  LEA  IACGlobals.ExtentTable(A2),A2;Get table base
  with   ExtentRec
  ADDQ.W #1,Edition(A2,D2.L)   ;Bump edition #
  MOVEA.L  EditionList(A2,D2.L),A3    ;Get handle to new
                chain head
  MOVE.L AØ,EditionList(A2,D2.L)    ;Point to new head of
                chain
  MOVE.L MstrInterestMsk(A2,D2.L),D6  ;Need to copy into
                editionHdr
  endwith    ;ExtentRec
  MOVEA.L  D7,AØ        ;Retrieve editionHdr handle
  MOVEA.L  (AØ),AØ        ;Hook list after new block
  MOVE.L A3,EditionHdr.NextEd(AØ);Set EditionList->next
  MOVE.L D6,EditionHdr.InterestMask(AØ) ;Remember who's
                interested
  MOVE.W WriteRec.formatCount(A4),D6  ;Get the passed
                formatCount
  MOVE.W D6,EditionHdr.NumFormats(AØ) ;set format count
  MOVEA.L  D7,AØ
  _HLock          ;Freeze new block
  MOVEA.L  (AØ),A3    ;Deref it…
OncePerFormat
  SUBI.W #4,D5      ;Knock down the format offset
  MOVE.L FmtDataBlock.dataSize(A1),DØ ;Get this block's
                size
  ADDI.L #4,DØ        ;Add room for the fmtcode
  MOVE.L DØ,D6        ;Set up count
  SUBQ.L #1,D6        ;Adjust for DBRA
  _NewHandle sys,clear    ;Try to allocate it
  BNE.S  WriteDataFailed     ;…Couldn't

  MOVE.L AØ,D6      ;Save handle to IAC data copy
  MOVE.L A1,D7        ;Save ptr to user data
  MOVEA.L  (AØ),AØ        ;Deref the IAC copy block
  with   FmtDataBlock
  MOVE.L dataSize(A1),DØ     ;Get this block's size
  MOVE.L formatCode(A1),(AØ)+  ;Copy the format code
  MOVEA.L  AØ,A1        ;Dest pointer for BlockMove
  MOVEA.L  D7,AØ      ;Source pointer for BlockMove
  ADDA.L #8,AØ        ;Skip hdr info
  _BlockMove
  MOVE.L D7,A1
  MOVE.L dataSize(A1),DØ     ;Get this block's size
  endwith    ;FmtDataBlock
  ADDQ.L #8,DØ        ;Allow for header
  ADD.L  DØ,A1    ;Reset A1 to next format block
  MOVE.L D6,AØ      ;Get handle to IAC data copy
  MOVE.L AØ,EditionHdr.EdInstances(A3,D5.L) ;Record handle in
EditionList
  TST.L  D5       ;Are we done?
  BNE.S  OncePerFormat     ;Do it again…
  MOVEA.L  WriteRec.theData(A4),AØ    ;Get the passed
```

```
                    theData
  _HUnlock        ;And unlock this puppy too
  MOVE.L ExtentRec.Edition(A2,D2.L),D4
  MOVE.L D4,WriteRec.edition(A4)    ;Update the edition
  MOVE.W #noErr,D0        ;We're copacetic at this
          point
WriteDataExit
  MOVEA.L  (SP)+,A1      ;restore dCtlStorage
  RTS           ;Bye-bye
WriteDataFailed
  MOVE.W #WriteFailed,D0      ;So solly
  BRA.S  WriteDataExit

  ENDPROC

;
; This routine is called by client to read data from the
driver
;
DriverReadData  PROC
    IMPORT FindExtentByDandHC,IsFormatAvailable

ReadRec    RECORD 0
Func   DS.W 1
docID    DS.L 1
slotID   DS.W 1
hatCheck DS.W 1
edition    DS.W 1
formatPref DS.L 3
formatCode DS.L 1
theData    DS.L 1
ReadRecSizeEQU *
    ENDR

  MOVEA.L  (A6),A2         ;Dereference globals handle
  MOVE.L ReadRec.docID(A4),D0  ;Get the passed docID
  MOVE.W ReadRec.hatCheck(A4),D1    ;Get the passed hatCheck
  JSR  FindExtentByDandHC     ;Get dependancy if it exists
  BNE  ReadDataExit       ;If subr plotzed get out. Otherwise,
  MOVEA.L  (A6),A2         ;Dereference globals handle
  LEA  IACGlobals.ExtentTable(A2),A3;Get table base
  MOVE.W ExtentRec.Edition(A3,D2.L),D4;Fetch latest stored
                        edition
  CMPI.W #-1,D4        ;If edition not -1
  BNE.S  LinkStillValid      ;…Link is OK
  MOVE.W #MissingLink,D0     ;…Else link was invalidated
  BRA  ReadDataExit      ;Go bye-bye
LinkStillValid
  CMP.W  ReadRec.edition(A4),D4  ;Has he seen my latest?
  BLE.S  FetchLatestEdition
  MOVE.W #NoNewerEdition,D0     ;Yes, he has…
  BRA  ReadDataExit
FetchLatestEdition
  MOVEA.L  ExtentRec.EditionList(A3,D2.L),A0;The handle for
                            latest ed'n
  MOVEA.L  (A0),A3         ;A3 -> latest ed'n
  _HLock          ;Freeze this puppy
  LEA  ReadRec.formatPref(A4),A2  ;Point to first format
  JSR  IsFormatAvailable   ;Is format 1 ok?
  BNE.S  RenderMe      ;If so, render it…
  LEA  4(A2),A2      ;Point to second format
  TST.W  (A2)        ;If no format 2…
  BEQ  ReadDataFailed
  JSR  IsFormatAvailable   ;Is format 2 ok?
  BNE.S  RenderMe      ;If so, render it…
  LEA  4(A2),A2      ;Point to third format
  TST.W  (A2)        ;If no format 3…
  BEQ  ReadDataFailed
  JSR  IsFormatAvailable   ;Is format 3 ok?
  BEQ  ReadDataFailed      ;If not, bail out…
RenderMe
  MOVE.L (A2),ReadRec.formatCode(A4)  ;Set format code
  MOVEA.L  (A6),A2         ;Dereference globals handle
  LEA  IACGlobals.ExtentTable(A2),A2;Get table base
  MOVEA.L  ExtentRec.EditionList(A2,D2.L),A0;The handle for
latest ed'n
  MOVE.L A0,-(SP)      ;Save for later
  _HUnlock        ;Dismiss it
  MOVE.L D7,A0         ;Put render handle in A0
  _GetHandleSize         ;How big are we?
  SUBQ.L #4,D0         ;Fudge size param for type code
  MOVE.L D0,D6         ;Save size param
  _HLock          ;Freeze the EdInstance
  MOVEA.L  (A0),A3         ;Deref it
  MOVEA.L  ReadRec.theData(A4),A0  ;Get output handle
  MOVE.L D6,D0         ;Get real size
  _SetHandleSize         ;Make enough room…
  MOVE.L A1,-(SP)      ;Stack dCtlEntry
  MOVEA.L  (A0),A1         ;Destination
  LEA  4(A3),A0      ;Source starts after formatCode
  MOVE.L D6,D0         ;Reload real size
  _BlockMove
  MOVE.L (SP)+,A1      ;Retrieve dCtlEntry
  MOVE.L D7,A0         ;Put render handle in A0
  _HUnlock        ;Let it go
;
; Now delete edition if no further interest
;
KillEditionLoop
  TST.L  (SP)        ;Valid address?
  BEQ.S  ReadDataOK      ;Exit if not..
  MOVEA.L  (SP),A0         ;Reload handle to edition header
  MOVEA.L  (A0),A3         ;A3 -> latest ed'n
  with   EditionHdr
  MOVE.L InterestMask(A3),D5   ;Get interest mask
  MOVE.W ReadRec.slotID(A4),D0    ;Get bit position
  BCLR.L D0,D5         ;Clear interest bit
  MOVE.L D5,InterestMask(A3)   ;Save updated mask
  BNE.S  ReadDataOK      ;Still interested parties, keep data
  MOVE.W NumFormats(A3),D7   ;Loop limit
  SUBQ.W #1,D7         ;Set up for DBRA
  LEA  EdInstances(A3),A2    ;Point to base of table
  endwith    ;EditionHdr
KillLoop
  MOVEA.L  (A2),A0         ;Handle to data
  _DisposHandle        ;Kill it
  LEA  4(A2),A2      ;Next entry
  DBRA D7,KillLoop
  MOVE.L EditionHdr.NextEd(A3),D5 ;Save Link
  MOVEA.L  (SP),A0         ;Reload header handle
  _DisposHandle        ;Kill it
  MOVE.L D5,(SP)         ;Replace handle with link
  BRA.S  KillEditionLoop     ;Try removing earlier edition

ReadDataOK
  MOVEA.L  (A6),A2         ;Dereference globals handle
  LEA  IACGlobals.ExtentTable(A2),A2;Get table base
  MOVE.L (SP)+,ExtentRec.EditionList(A2,D2.L)        ;Relink
handle for latest ed'n
  MOVE.W #0,D0         ;Indicate exit OK
ReadDataExit
  RTS

ReadDataFailed
  MOVE.W #ReadFailed,D0      ;So solly
  BRA.S  ReadDataExit

  ENDPROC

;
; Unused entry points that must be defined for the header's
jump table
;
DriverCtl  PROC
    EXPORT DriverDone
DriverDone
  MOVEQ  #0,D0         ;Everything's cool
  RTS           ;Return to sender
```

```
  ENDPROC

;
;Many years later...the Subroutines!!
;
;Returns D0.W = slot_ID
;  A3,D1.W pointing to available entry in docID list
FindSlotID PROC
  MOVEA.L  (A6),A2        ;Dereference global handle
  LEA IACGlobals.docIDs(A2),A3 ;Point to list of docIDs
  MOVEQ  #-4,D1       ;Starting index of zero
  MOVEQ  #0,D3        ;Starting slot_ID
SlotIDLoop
  ADD.W  #4,D1        ;Bump to next index
  ADD.W  #1,D3        ;Bump counter
  CMP.W  #NumOfDocs+1,D3     ;Are we done yet?
  BEQ.S  SlotIDFail     ;If so, we can't create slot_ID
  CMP.L  (A3,D1.W),D0      ;Compare it to our docID
  BEQ.S  StuffSlotID    ;Exit if we're done
  TST.L  (A3,D1.W)      ;See if open slot
  BNE.S  SlotIDLoop     ;If not, try again
StuffSlotID
  MOVE.W D3,D0        ;Move to result register
SlotExit
  TST.W  D0
  RTS          ;Return to sender
SlotIDFail
  MOVE.W #NoMoreSlots,D0     ;Custom OSErr
  BRA.S  SlotExit

  ENDPROC

;
;Looks for dep in extent table by matching docID and hatCheck
;ASSUMES THE FOLLOWING
; D0 = docID
; D1 = hatCheck
; A6 = globals handle [always]
;RETURNS
; D0 = noErr OR NoSuchDep
; D2 = desired offset
;We merrily destroy A2 and A3 during this routine
;
FindExtentByDandHC  PROC
  MOVEA.L  (A6),A2
  LEA  IACGlobals.DocIDs(A2),A3 ;Point past end
  LEA  IACGlobals.ExtentTable(A2),A2;Point to extent table
  MOVEQ  #0,D2        ;Clear our offset
FEBDAHCLoop
  CMP.L  ExtentRec.docID(A2),D0  ;Check this extent's docID
  BNE.S  SkipToNextExtent    ;If wrong, skip to next
  CMP.W  ExtentRec.hatCheck(A2),D1 ;Check this extent's docID
  BNE.S  SkipToNextExtent    ;If wrong, skip to next
  MOVE.W #noErr,D0       ;Nothing is wrong
  BRA.S  FEBDAHCExit     ;Get out.

SkipToNextExtent
  ADD.L  #ExtentRec.ExtentSize,D2 ;Bump offset
  ADDA.L #ExtentRec.ExtentSize,A2 ;Point to next record
  CMPA.L A2,A3         ;Are we done?
  BGT.S  FEBDAHCLoop     ;If not, look again

FEBDAHCFail
  MOVE.W #NoSuchDep,D0     ;Return custom OSErr

FEBDAHCExit
  TST.W  D0
  RTS          ;Bye-bye

  ENDPROC

;
;This routine wipes out all data and formats thereof for a
single extent
;ASSUMES
; A0 = current EditionList
;We merrily destroy A3,D0,and D5 (A2 restored from A6)
;RETURNS
; Nothing in particular
WipeExtentData  PROC
  with IACGlobals
  with ExtentRec

  MOVE.L A0,D0      ;Test handle to Edition List
  BEQ.S  DoneExtent    ;If none, do nothing
  MOVEA.L  D0,A3       ;Keep Edition List handle

NextEdition
  MOVEA.L  A3,A0       ;Get Edition List handle
  _HLock         ;Lock the Edition List handle
  MOVEA.L  (A3),A0       ;Get pointer to Edition List

  with EditionHdr
  LEA  EdInstances(A0),A2  ;Point to instance data
  MOVE.W NumFormats(A0),D5 ;Get counter of formats

KillFormat
  MOVEA.L  (A2),A0       ;Get the handle to the data
  _DisposHandle     ;Get rid of it
  LEA  4(A2),A2   ;Point to next instance
  SUBQ.W #1,D5       ;Decrement the counter
  BNE.S  KillFormat    ;And do it again

  MOVEA.L  (A3),A0       ;Dereference the Edition List handle
  MOVE.L NextEd(A0),D5  ;Get handle to next Edition
  MOVEA.L  A3,A0       ;Restore handle
  _HUnlock      ;Unlock it
  _DisposHandle     ;Get rid of it
  TST.L  D5     ;Is there a next Edition?
  BEQ.S  DoneExtent    ;If not, go to next extent
  MOVEA.L  D5,A3       ;Otherwise move the handle
  BRA.S  NextEdition  ;And loop

  endwith    ;EditionHdr
  endwith    ;ExtentRec
  endwith    ;IACGlobals

DoneExtent
  MOVEA.L  (A6),A2       ;Restore
  RTS

  ENDPROC

;
;Looks for an dep in the extent table by matching the docID
and hatCheck
;ASSUMES THE FOLLOWING
; A2 = pointer to format(integer)
; A3 = pointer to Edition record
;RETURNS
; D7 = NIL OR handle to EdInstance
;We do not touch A2,A4 during this routine
;We merrily destroy A0,D4,D5 during this routine
;
IsFormatAvailable PROC
  MOVE.W EditionHdr.NumFormats(A3),D5 ;Get number of
              formats available
  MOVE.L (A2),D4         ;Hold desired fmt code
  SUBI.W #1,D5
  LSL.W  #2,D5         ;Make it an offset
  MOVE.L #0,D7         ;Set return to NIL
FormatCheck
  BLT.S  FormatCheckDone     ;D5 is now zero
  MOVEA.L  EditionHdr.EdInstances(A3,D5.W),A0        ;Move
handle to EdInstance
  MOVE.L A0,D7         ;Save in case it's good
  MOVEA.L  (A0),A0         ;Deref
```

```
  CMP.L  (A0),D4        ;Are the formats the same??
  BEQ.S  FormatCheckDone     ;If so, leave
  SUBI.W #4,D5        ;Check next one back…
  BRA.S  FormatCheck
FormatCheckDone
  TST.L  D7       ;Set the condition code first
  RTS

  ENDPROC

  END
```

Listing SAWSINIT.a

```
;*****************************************************************
;*****The SAWS Inter-Application Communication Driver Loader
;*****Written with blazing speed 6-7/88 by Paul F. Snively
;*****  With one HELL of a lotta help from Frank Alviani
;*****************************************************************
;
;Modification History:
;6/19/88 First Draft-Paul F. Snively
;7/10/88 Hopefully last draft-this SHOULD work-
;Paul F. Snively
;8/21/88 Now searches for open slot from end of table-
;Frank Alviani
; (Algorithm from Pete Helme, Apple MACDTS)
;
;Basically what this puppy does is to assume that there's a
;DRVR resource lying around that happens to be named ".IAC"
;and, if there is, it loads it into the
;System Heap and opens it.  Simple, huh?
;

    INCLUDE  'Traps.a'
    INCLUDE  'QuickEqu.a'
    INCLUDE  'SysEqu.a'
    INCLUDE  'ToolEqu.a'
;   STRING ASIS

successID  EQU  128
failureID  EQU  129

SAWSINIT:  PROC

  IMPORT ShowINIT

Frame  RECORD 4,DECR
return DS.L 1
A6Link DS.L 1
theID  DS.W 1
theType  DS.L 1
name DS.B 256
fSize  EQU *
  ENDR

  LINK A6,#Frame.fSize     ;space for locals...
; Find an open slot in the driver table and load into that
  MOVE.W UnitNtryCnt,D2      ;How many slots?
  SUBQ.W #1,D2        ;Adjust
  MOVE.W D2,D1        ;Set up offset
  LSL.W  #2,D1
  MOVEA.L  UTableBase,A0     ;Where's the table?
SrchLoop
  TST.L  0(A0,D1.W)      ;Available?
  BEQ.S  GotSlot         ;Yup...
  SUBQ.L #4,D1        ;Drop Offset
  SUBQ.W #1,D2        ;Drop slot ID
  CMPI.L #39,D2       ;At bottom limit?
  BGT.S  SrchLoop
  BRA.S  BadNews    ;No open slots in the inn
;Get resource by name
GotSlot
  SUBQ.W #4,A7        ;Space for handle
  MOVE.L #$44525652,-(A7)    ;'DRVR'
  PEA  DriverName
  _GetNamedResource
  MOVE.W ResErr,D0       ;Get it?
  BNE.S  BadNews
  MOVE.L (A7)+,D7   ;Was there enough memory?
  BEQ.S  BadNews
;Change ID to open slot
  MOVE.L D7,-(A7)
  PEA  Frame.theID(A6)     ;ID
  PEA  Frame.theType(A6)   ;theType
  PEA  Frame.name(A6)      ;name
  _GetResInfo
  MOVE.L D7,-(A7)
  MOVE.W D2,-(A7)
  PEA  Frame.name(A6)      ;name
  _SetResInfo
;Open the driver!
  CLR.W  -(A7)        ;Room for refnum
  PEA  DriverName      ;Point to driver name
  _OpenDeskAcc         ;Open it
  MOVE.W (A7)+,D1      ;Pop refnum
;Ensure Driver undisturbed
  MOVE.L D7,-(A7)
  _DetachResource
;Restore previous slot in file
  MOVE.L #$44525652,-(A7)    ;'DRVR'
  PEA  DriverName
  _GetNamedResource
  MOVE.L D7,-(A7)
  MOVE.W Frame.theID(A6),D0
  MOVE.W D0,-(A7)
  MOVE.L #0,-(A7)      ;leave alone name
  _SetResInfo
  MOVE.W #successID,-(SP)    ;Stack success ICN# ID
ShowICON:
  MOVE.W #-1,-(SP)     ;Use standard pixel offset
  MOVE.L (SP)+,D0   ;TEMPORARY! POP OFF PARMS
; JSR  ShowINIT ;Tell user that driver installed
  UNLK A6
  RTS          ;And that's all, folks!
BadNews:
  MOVE.W #failureID,-(SP)    ;Tell user we failed
        miserably
  BRA.S  ShowICON      ;And leave

DriverName:
  DC.B '.IAC'     ;The name of our driver

  ENDPROC

  END
```

Listing SAWSINIT.r

```
/*The Resource Description File for the IAC Driver
    Frank Alviani - 8/88
*/

include $$Shell("hlxEtc") "SAWSdrvr.lnk" 'DRVR' (31)
  as 'DRVR' (31,".IAC",sysheap,nonpurgeable);        /* get
DRVR resource */
include $$Shell("hlxEtc") "SAWSINIT";     /* get INIT resource
*/

/* ICN# s will go here asap */
```

C Workshop

MenuStuff for Handling Menus

Mark Shirley
Paris, France
MacTutor Vol. 4 No. 10

MenuStuff

Menus are an essential part of the Macintosh interface, and the resource formats for creating menus and the ToolBox routines for implementing them and reacting to user choices are some of the most straight forward areas of Macintosh programming. But menus have two faces : the logical aspect, and what I will call the "housekeeping" aspect. Reacting to choice i,j by selecting font F is an example of the logical aspect; knowing that item j in menu i must be checked when chosen and that items j-m..j-1 and items j+1..j+n must be unchecked (to say nothing of the repercussions this might have on the display characteristics of Font Size Menu Z) is an example of the housekeeping aspect. This latter aspect is neither conceptually difficult nor algorithmically rewarding, but it generates an awful lot of extra code which is essentially similar from one program to another yet usually sufficiently program-specific that the routines end up being rewritten every time. Furthermore, the algorithmic triteness creates a paradox : programmer disinterest invites bugs, and it is usually late in the development cycle before these "visual feedback" features work correctly, whereas more difficult parts of the program, by capturing the programmer's interest, frequently are bug-free (sic) much earlier.

My solution to this problem(contrary to Beethoven, if you will excuse the presumption, I hate writing variations!), has been to :

1) list the different ways menu items behave and extract common types,

2) devise a custom resource format that codes the menu's "housekeeping" behavior; this custom resource complements (without modifying or replacing) the standard menu resource,

3) write a set of routines which, driven by the custom resource, carry out all housekeeping activities for the application program in a completely transparent manner, i.e., the program needs only call the front end after MenuSelect and then carry out the logical aspects of choice i,j in blissful ignorance of the mechanico-visual aspects of the User choice.

Obviously my menu types are not exhaustive. It seems that almost every new program that hits the market finds a new variant on the menu theme. Nevertheless the basic types are covered and new types are easy to create. The types work equally well whether the menu be straight pull-down, hierarchical, or pop-up (and presumably tear-off); dynamic item addition/suppression are supported (though with certain restrictions on the types); a mechanism is provided for storing/restoring a snapshot of a given menu state; dynamically allocated zones containing runtime information can be resource-triggered; and finally PROC type resources can be associated with menus and transparently executed on menu selection, thereby providing a mechanism for interactions between menus.

Excuse the drum-beating : it's really quite straightforward. There are no neat ToolBox tricks to be learned from this article, its only purpose is to relieve you of some of the drudge-work of menu handling.

The sources in this article are written in Consulair Mac C (having started it a long time ago, I updated in Mac C; it will be last thing I write in Mac C). Adapting it to the C language of your choice should be easy, insofar as no specific features are used (other than 32 bit int's).

Just as menu housekeeping in an application program takes up an amount of space disproportionate to its conceptual interest, so these routines are rather long. In order to fit it into the space graciously allotted by the Editor (let us not dwell on his displeasure with the original size), I had to cut out quite a bit of code, and compact the remaining code to the detriment of its legibility. The source available with this issue contains the full length code along with a very thorough test program that allows complete testing of menus before integrating them into the final application. This is very important, because the custom nature of the resource makes coding it very error-prone, and the source code presented has had the data-validation part amputated and the error reporting routine severely watered down. In the perfect leisure world of my fantasy, rather than using RMaker to create the custom resource it would be generated by an interactive utility.

Name Calling

Not surprisingly, the key data structure is called MenuStuff, and this structure will be referred to as MS (that these also happen to be my initials is of course pure coincidence). Its complete description may be found in "MS.h".

Item Types

The basic item types recognized by MS routines are the following :

1) Normal : straight-Joe items requiring no special processing.

2) Check Toggled : an individual item which is either checked or not checked, and toggles between these states with each successive selection. Example : Gremlin Alert in QUED.

3) Text Toggled : an individual item whose text exists in two variants; successive selections toggle the display of these alternative text variants. Example : a Hide/Show item.

4) Enabled Pair : a pair of menu items one of which one is enabled and the other disabled at any given time. Selecting the enabled member disables the selected member and enables the other member of the pair. Example : Open/Close items in MacWrite.

5) Checked Range : a set of items for which one and only one element must be checked at any given time. Example : a Font Size menu.

6) Cumulative Range with Reset : a set of items with one singular element (the parent) and one or more non-singular elements (members); at any given time at least one element must be checked. Several members may be concurrently checked, but the parent and members are never checked concurrently; selecting a member unchecks the parent, and unchecking the last checked member checks the parent (no, this is not a chess journal, you did pick up the right magazine). Individually, members behave like check toggled items. Archetypical example : the Style Menu.

7) Enabled Set : a set of items comprised of two disjoint subsets. Each subset contains a singular element (the parent) and one or more non-singular elements (members); at any given time all elements of one of the disjoint subsets are enabled and all members of the other subset are disabled. Selecting the enabled parent of the enabled subset disables all elements of the selected subset and enables all elements of the other disjoint subset. Individually, the member elements behave like normal items.

8) Text Toggled Set : a set of items with one singular element (the parent) and one or more non-singular elements (members); each element's display text exists in two variants, and successive selections of the parent toggle the display of these alternative text variants for all elements in the set. Variant display is coherent, i.e, at any given time the same variant level is displayed by all elements of the set. Individually, member elements behave like normal types.

9) Disabled Menu Lines : self-explanatory; used to "parse" set membership when set membership is defaulted in the MS resource, otherwise ignored by MS routines (more on this point later).

I am not trying to dictate menu behavior, I am describing the predefined types this version of MS knows how to handle. Any type of behavior differing from the predefined types may either be tagged as Normal (hence ignored by MS routines and fielded by the application), or else new types may be added to the MS routines.

Functional Overview

Menu creation per se is totally independent of MS. Once menus are created, if the menu contains items the programmer wants MS routines to handle, a complementary MS resource must be created using RMaker or equivalent. At this point linking the test program (MST.Rel) with the menu and MS resources will immediately show not only what the menus look like, but also how they react to user choices. Not a line of application code has so far been written, yet the menu interface is a faithful presage of the real thing.

In the application, a call to MSMake should be followed by a call to MSInsert. Then, in the doMenu section of the application, before the Menu/ItemNumber Switch, a call to MSDispatch should be inserted. To delete a menu from the MenuBar without disposing of memory structures (both standard and MS), the application should call the ToolBox DeleteMenu routine. To dispose of associated memory structures (again, both standard and MS) the application should call MSDispose. To add/delete items to/from existing menus MSInsMenuItem/MSDelMenuItem should be called. See below for snapshots and inter-menu interactions.

In its present form MS routines do not act on MenuBars, but adding a front end to MSMake, MSInsert, and MSDispose is trivial. Actually MS itself is trivial, and the the point of this whole article is to prove the usefulness of coordinated triteness.

Underlying Mechanics

More on name calling : up until now I have used MS for MenuStuff and consistently qualified it with either "routines" or "resources". The main MS resources type is 'MST ' (MST blank); therefore I will henceforth use MST to designate the resource and MS to generically designate MS routines.

MST is a non-purgeable, variable length resource that MS never rewrites to disk. As you will see shortly, menu states can be faithfully memorized and restored using a small, masked flag structure. MST is composed of constant header fields affecting the entire menu, a variable size Menu Verb Table (MVT) coding behavior at the item level, and an optional 'PROC' function parameter zone following the MVT (more on this later).

Each MVT entry is a MenuVerb structure, composed of an opCode component and a 5-element argument table (argT) component. OpCodes and argTs are typedef'ed to byte, which is adequate for the predefined types but may be changed if longer argTs should be necessary for new types. MVT[0] contains bit flags (referred to as Menu Flags) which affect the way menus are set up by MSMake, and govern certain aspects of PROC function behavior. The opCode identifies the item type, and the argTs supply positional information necessary to implement the type's behavior. Not all types use all the argTs, and argT[4] (the last one) is never used by any predefined type; it pads to word length boundary, and may be used for new types, or it can hold any item information the calling application might wish to store there. In general, argT[0] through argT[4] correspond to "first", "last", "parent" and "other" positional parameters, but in the case of TextToggled items, for example, "other" (argT[4]) is the index of the first text variant in the associated STR# resource (the second variant is other+1). These positional arguments may be ex-

pressed either as absolute itemNumbers or may be calculated by MS based on the previous and the next disabled line positions in the menu. Using direct references allows interspersing different members of a given group anywhere in the menu (as long as there are no intervening types with the same opCode), whereas using disabled line defaulting makes coding the resources a lot easier. A concrete example : for an Enabled Set, first and last are the position of the first and last element of THIS disjoint set, parent is the master item controlling THIS disjoint set's enabling/ disabling, and other is the itemNumber of the OTHER disjoint set's parent.

Another essential part of the MST resource is the ToggleFlag component (not to be confused with the menu flags in MVT[0]). In the present version ToggleFlag is typedef'ed to MenuVerb, giving a 48 bit field, but this can be increased to handle more items (but have mercy on the poor User!). Every time an item different from Normal or Disabled Line is passed to MSDispatch, the associated flag is toggled from its previous state. This new value determines the transition processing. For example, toggling a text toggled item causes the other+ToggleFlag[i] variant to be displayed. Certain transitions, such as a 1->0 on a Checked Range are not allowed and the toggle flag is reset to its original value (otherwise there would be no checked member in the range). With one exception (Expanded Ranges, see below), the initial state of ToggleFlags must be CORRECTLY coded into the resource; bits are numbered from left to right to allow for compatibility should ToggleFlags need to be lengthened. Bit zero is not used.

Menu flags, occupying the first MVT entry, control the overall set up of the menu. There is a flag indicating if AddResMenu should be called, and if it is set, the field sysRes in the constant header should contain a resource type. Since the number of resource items cannot be determined at resource creation time, it is impossible to code MVT entries for these items. To solve this problem, another flag enables MVT expansion, and yet another enables ToggleFlag set up and initial menu checking. In the present version only Checked Ranges may be expanded using this mechanism, and this is adequate for most applications. Two fields in the constant header govern this expansion : staticCount and allocCount. StaticCount is the itemNumber of the last MVT explicitly coded, and allocCount is the total number of entries allocated in the MVT. Entries beyond staticCount should be coded as 6 zero bytes. After the resource is added, the staticCount entry is copied from the staticCount to the dynamicCount-staticCount last entries (dynamicCount is determined by a CountMItem call). If the number of added items should exceed allocCount, the menu is trimmed to allocCount. A flag indicating whether the user should be warned of this trimming is set aside but the warning is not implemented in this version. Increasing the maximum toggle flag "addressing" capacity only adds 1 bit of

MenuVerb Parameters

opCode	argT[0] (first)	argT[1] (last)	argT[2] (parent)	argT[3] (other)	argT[4] (unused)	ToggleFlag Transition (result after toggling)
Normal	unused	unused	unused	unused		•Not toggled, not processed;
CheckToggled	unused	unused	unused	unused		•1-> Chk, 0->Unchk.
TextToggled	unused	unused	unused	default STR# Indx (if NULL, patched by MSMake)		•Display STR# string indexed by (other+TogFlg).
Enabled Pair	unused	unused	unused	other member,defaults to : zero->NEXT,<zero->PREV		•1->ignored; 0-> Disable this item, Enable other item.
Checked Set	first item # (defaults to first item in linegroup)	last item # (defaults to last item in linegroup)	unused	for Exp Rng, # of member to be initially checked (defaults to 1 or MIVT defVal)		•1->Chk, unchk other mbrs 0->Not allowed.
Cumulative Range with Reset	Idem	Idem	parent item # (defaults to first item in linegroup)	unused		•memb: 1->chk, if parent chked unchk prnt; 0->unchk, if no other mbr chked chk prnt ; parent: 1->chk prnt; unchk mbrs; 0->retoggled.
Enabled Set	Idem	Idem	Idem	other Set parent, defaults to : zero->first of NEXT line grp, <zero->first of PREV line grp;		•member: retoggled; parent : 1->ignored; 0->disable this set,enable other set;
TextToggled Set	Idem	Idem		default STR# Indx (if NULL, patched by MSMake)		•member : retoggled; parent: group display text set to STR# string indexed by (other+TogFlg)
Disabled Menu Line	unused	unused	unused	unused		•ignored; only significant for default line parsing;

overhead per item, so it is easy to allow for lots of fonts, or whatever other resources the application needs. Only the size of menus actually concerned with this resource will be increased beyond this 1 bit overhead.

The Expanded Check Range mechanism, though strictly necessary only for resource additions, can also be used to expand any Checked Range. The staticCount entry must have a Check Range opCode, and if the first argT is less than or equal to zero, the StaticCount entry will be copied into the dynamicCount-staticCount last entries, as explained above, and the "first" argT will be set to staticCount and the "last" argT will be set to dynamicCount. If however, the first argT is different from zero, the staticCount entry will still be copied, but from "first" to "last" only, without modification.

There are three ways to specify which entry in the Expanded Range should initially be checked and have its toggle flag set :

• argT[4] ("other") is not normally used in Checked Range types; if the Expanded Range flag is set, however, and this argT is non-zero, it will be interpreted as the default item relative to staticCount, e.g., 3 would mean StaticCount + 2,

• if this argT is zero, then if the MST has an associated MIVT (Menu Item Value Table), and if the table contains an entry whose value is equal to the MIVTdefVal field, then the position of this value in the table gives the item to be checked,

• if these schemes fail, item number staticCount is checked and set.

Special Features

In order for the application to dynamically add items to an installed MST menu, extra space must follow the staticCount entry in the MVT, i.e, staticCount must be strictly less than allocCount, and as items are added/deleted, dynamicCount must never exceed allocCount. Notice that MSInsMenuItem does not exit via MSErrExit on error, it returns a negative error code instead. Items are added by passing a menuID, an item string pointer (which is passed directly directly to the ToolBox InsMenuItem routine, so it may include metacharacters), an after and a MenuVerb parameter. If the after parameter is non-null, it is taken as relative to StaticCount-1, e.g., both zero and 1 give staticCount and 5 gives staticCount + 4; out of range values are clipped to either staticCount or dynamicCount. Normal return values give the absolute itemNumber of the added item. Toggle flags are not set and the menu is neither checked nor enabled as a result of calling MSInsMenuItem, so the addition of singular elements should be followed by a call to MSDispatch, passing it the absolute itemNumber returned by MSInsMenuItem. TextToggled and TextToggled Sets MAY NOT BE ADDED IN THE PRESENT VERSION. Different groups should either be separated by a disabled line insertion, or exact argT parameters MUST be supplied, though interesting effects can be obtained by mixing different types, so caveat emptor.

The MIVT/PROC mechanisms are extremely useful. Though they can be used independently, they complement each other and the dynamic menu addition mechanism to provide a powerful way of writing resource-driven processing that is independent of the main application.

A menu flag governs the allocation of an MIVT (Menu Item Value Table). The table consists of allocCount+1 longs allocated by NewHandle and initially set to MIVTnullVal. A handle to the table is stored in the MST. Another flag determines whether this table will be left alone or whether a PROC resource, whose ID is stored in procID[0], will be called to set it up. In either case a later call to MSGetMIVT, passing menuID and itemNumber, retrieves the entry.

A second PROC may be associated with the menu by storing its ID in procID[1]; this second PROC will automatically be called every time MSDispatch is entered. If the appropriate menu flag is set, the short integer following the MVT contains the number of shorts needed to store any additional parameters the second PROC might need to carry out its task (such as the ID of a or the secondary menus the selected menu might interact with).

Using PROC parameters allows for reusable code. Appropriate functions can be written and stored, and used by different applications. On the other hand, you do not have to use PROC functions. In addition to the blessed menuID, MSMake takes two additional parameters which will override the PROC IDs: they should be function addresses set up to receive the standard PROC parameters. These function addresses will be stored in the procID table of MST, and appropriate menu flags will be set by MSMake to indicate that the "PROC" functions are application functions and not really PROC resources.

The test files included contain three sample PROCs. PROC ID 1 calculates the font IDs for installed fonts, and stores them in the Font menu MIVT. PROC ID 3 scans the text of the size menu and converts the first numeric sequence it finds to a long integer and stores it in the Size menu MIVT. PROC ID 2 is associated with the Font menu, and is called every time this menu is selected. It retrieves the font number for the selected item from the Font MIVT, gets the ID of the Size menu from the optional PROC parameters following the MVT, and then scans the Size menu entries, retrieving the font size from its MIVT. Any size entry that matches a RealFont gets its text outlined. The calling application is never aware of any of this visual processing, and simply retrieves font IDs and numeric sizes for logical actions associated with the menu choice without having to declare tables, make conversions, etc.

The PROC mechanism and the dynamic menu addition features provide a convenient solution to the problem of displaying open window titles in a menu, with the front window item typically checked. When the user chooses a window menu item, the corresponding window is brought to the front. This is a very common feature of most multiple window programs.

Initially no windows are open, and no window items appear in the "Window" menu. When a window is opened, MSInsMenuItem should be called, passing a menuID,a Checked Range type Menu Verb having zero first and last parameters (the staticCount item is assumed for convenience to be a disabled menu line), the Window Title string pointer, and either zero or a large number as the after parameter. Then the windowPtr should be stored in

the MIVT, in the entry whose index is the itemNumber returned by MSInsMenuItem (the short version of MS presented here lacks the MSSetMIVT and MSFindMIVT functions, but they are easy to code, and are present in the longer version included in the source disquette). Next a call to MSDispatch will check this item.

When a window is closed, find its windowPtr in the MIVT, and the corresponding index is the itemNumber to delete by calling MSDelMenuItem.

The Toggle Flags provide the means by which menu states may be memorized throughout without ever changing the main MST disk resource. The MSAdjTog function takes a menuID and two ToggleFlag pointers as parameters. The first ToggleFlag pointer is a ToggleFlag, and the second is a ToggleFlag mask. For every bit for which the corresponding mask bit is not zero, if the corresponding installed Toggle bit is different from the passed Toggle bit, MSDispatch is called for this item, effectively setting the menu to the snapshot state implicitly contained in the passed ToggleFlag/Mask pair. Processes whose states are reflected in menus need only store the appropriate masked toggle flags before deactivation, and call MSAdjTog passing this stored information when reactivated.

A special resource, of type 'MST1', consists of just such a pair. These resources are optional, and their existence is determined by the state of one of the Menu Flags in MVT[0] (see above). When menus are originally set up by MSMake, if this flag is set , the 'MST1' resource will automatically be retrieved and MSAdjTog called. The application may update this resource and re-write it to disk by calling MSAdjPREV, which examines the mask contained in the appropriate 'MST1' resource and updates the ToggleFlag part of the disk resource according to the value of installed Toggle bits.

Extensions

An easy extension would allow the application to set up but never insert a special MST menu the user would never see. This invisible menu would serve to code multiple menu interactions provoked by non-menu events (program choices, state changes, etc.). Code for new types would need to be included in MSDispatch1 & 2. These new types would never actually be accessed by User menu choices : the applications simulates MSDispatch calls on the dummy menu. The dummy menu may contain opCodes for which the argTs would be interpreted differently . For example, to implement a mass disabling, suppose new type for which :

- [first] item number action applied to
- [last] item number action applied to
- [parent] is the singular element for the group
- [other] is menuID to other alternate set (forces menuIDs to 0-255, or need to change argT type to short)
- [(currently unused) argT[4]] is the number of entries following this one.

The MSDispatch code could be set for example to enable/disable (depending on the dummy menus own toggle flag transition), the specified range of the specified menu, for each item verb in the list. Therefore a parenthetical call to MSDispatch with the dummy i,j could trigger enabling/disabling over a multiple-menu set of items, independently of their primary MS coding.

A far less obvious extension would be to develop an MST like structure controlling DITLs and other controls. Hmm...

A better mousetrap ?

Obviously for small programs using few menus, MS is shooting mice with elephant guns. For large programs, relying extensively on menus, MS offers a significant savings in programming time, it provides a way of rapidly visualizing the functioning menu interface, and may be easily adapted to provide for non-menu generated events having multiple-menu visual display repercussions. But is MST fun ?

Listing: MS.h

```
/*MS.h = metavariables and typedefs for MS routines*/
#include "stdio.h"
#include "MacDefs.h"
#include "Resource.h"
#include "Menu.h"
// MS.h = MenuStuff metavariables
/*_____ Menu Verb Codes _____*/
#define  _MVNORMAL  0      /* NORMAL ITEM */
#define  _MVCHKTOG  1      /* CHECK TOGGLED */
#define  _MVTEXTOG  2      /* TEXT TOGGLED */
#define  _MVENABPR  3      /* ENABLED PAIR */
#define  _MVNZEXRG  4      /* NON ZERO EXCL RANGE     */
#define  _MVCMRGRS  5      /* CUMULATIVE RNG WITH RESET*/
#define  _MVENBSET  6      /* ENABLED  SET */
#define  _MVTTGSET  7      /* TEXT TOGGLED SET        */
#define  _MVMENLIN  -128 /* DISABLED LINE             */

/*_____ Menu Flags _____*/
#define  _MFSYSLOD  0      /* LOAD SYSTEM RESOURCE    */
#define  _MFERRTRM  1      /* ALERT USER TO TRIM      */
#define  _MFEXPRNG  2      /* EXPAND RANGE            */
#define  _MFALOCTB  3      /* ALLOCATE MIVT TABLE     */
#define  _MFBLDTAB  4      /* BUILD MIVT TABLE        */
#define  _MFSETXRG  5      /* SET EXP. RANGE TOGGLEFLG &
MENU*/
#define  _MFSETPRE  6      /* SET TO PREVious STATE   */
#define  _MFSUBMEN  7      /* IS A SUB-MENU           */
#define  _MFHASSTR  8      /* HAS TXTTOG STR# RESOURCE*/
#define  _MFPRCPAR  9      /* PROC PARAMETERS FOLLOW MVT*/
#define  _MFPRCFLG  14   /* USED INTERNALLY FOR PROC FUNCTS*/

/*_____ Resource Strings _____*/
#define  _MSMSTTYP  'MST ' /* MENUSTUFF TYPE          */
#define  _MSSTRTYP  'STR#' /* TXTTOG  STR# RESOURCE   TYPE*/
#define  _MSPRCTYP  'PROC' /* MIVT FUNCTION TYPE      */
#define  _MSPRVTYP  'PREV' /* RES FOR RESET TO PREV STATE*/

/*_____TypeDefs _____*/
typedef  char   MVOpCode;
typedef  char   MVArgT;
#define  _MVARGDIM    5

typedef  struct
{ MVOpCode opCode;
  MVArgT   argT[_MVARGDIM];
} MenuVerb;

typedef  MenuVerb ToggleFlag;
#define  _TFBITSZE  (sizeof(ToggleFlag)*8-1)

typedef struct
{
  MenuHandle   MH;
```

```
  long     sysRes;
  long     **MIVTH;
  long     procID[2];
  long     MIVTdefVal;
  long     MIVTnullVal;
  short      staticCount;
  short      allocCount;
  short      beforeID;
  short      ttgStrID;
  ToggleFlag   TF;
  MenuVerb   MVT[_TFBITSZE+1];
}MenuStuff;
#define    _MSFIXLEN    42
typedef MenuStuff   *MSPtr;
typedef MSPtr     *MSHdl;

/*_____________ Non-int Externals _____________________*/
extern MSHdl    MSGetMSH();
```

Listing MS.C

```
/*MS.C = source code for MenuStuff Routines */
#include "MS.h"

int MSMake(menuID, MIVTFct1,MIVTFct2)
short          menuID;
  int            MIVTFct1;
  int            MIVTFct2;
( /*Main Set Up routine - must be called before using menu*/
  MSHdl      theMSH;
  MSPtr      theMSP;
  MenuVerb theMV, theMF;
  Handle   theHdl;
  long     defVal;
  int        i,j;
  short      dynamCount,staticCount,allocCount;
  MVArgT   defPos;
  MVOpCode opCode;
  short      progNum=01;

  HLock((theMSH=MSGetMSH(menuID,progNum)));
  theMSP=*theMSH;
  theMF=*theMSP->MVT;
  /* Get Menus */
  if (! (theMSP->MH=GetMenu(menuID)) )
    MSErrExit(progNum,02,menuID,0);
  /* Load System resources */
  if (BitTst(&theMF, _MFSYSLOD))
    AddResMenu(theMSP->MH,theMSP->sysRes);
  /* Trim to allocCount */
  dynamCount=CountMItems(theMSP->MH);
  staticCount=theMSP->staticCount;
  allocCount=theMSP->allocCount;
  while (dynamCount > allocCount)
    { DelMenuItem(theMSP->MH,dynamCount);dynamCount-;}
  /* Minimal validation */
  if (staticCount > allocCount)
    MSErrExit(progNum,18,menuID,0);
  /* If TEXT TOG, check STR# & set Indx */
  if (BitTst(&theMF, _MFHASSTR))
  { if (! GetResource(_MSSTRTYP,theMSP->ttgStrID) )
      MSErrExit(progNum,07,menuID,0);
      /*  */
    j=0;
    for (i=1;i <= allocCount;i++)
    { opCode=theMSP->MVT[i].opCode;
      if (opCode == _MVTEXTOG || opCode == _MVTTGSET)
      { if (theMSP->MVT[i].argT[3] <= 0)
          (theMSP->MVT[i].argT[3]=j*2+1;j++;)
        /* Set TXTOG items to TF state */
        theMV.opCode=theMSP->MVT[i].opCode;
        theMSP->MVT[i].opCode=_MVTEXTOG;
        MSTogBit(menuID,i);
        MSDispatch1(theMSP,menuID,i);
        theMSP->MVT[i].opCode=theMV.opCode;
  } } }
  /* Expand  Range for added resources */
  if (BitTst(&theMF, _MFEXPRNG))
  { theMV=theMSP->MVT[staticCount];
      if (theMV.opCode != _MVNZEXRG)
        MSErrExit(progNum,09,menuID,staticCount);
      if (theMV.argT[0] <= 0)
      (   theMV.argT[0]=staticCount;
          theMV.argT[1]=dynamCount;
          if (theMV.argT[3])
            theMV.argT[3]+=staticCount-1;
      )
      for (i=theMV.argT[0];i <= theMV.argT[1];i++)
        theMSP->MVT[i]=theMV;
  }
  /* Allocate MIVT (Menu Item Values Table*/
  if (BitTst(&theMF, _MFALOCTB))
      if(!(theMSP->MIVTH=
          (long **)NewHandle((allocCount+1)*sizeof(long))))
            MSErrExit(progNum,03,menuID,0);
      else for (i=0;i <= allocCount;i++)
              *((*theMSP->MIVTH)+i)=theMSP->MIVTnullVal;
  /* MIVT Prep */
  if (MIVTFct1)
  { theMSP->procID[0]=MIVTFct1;
    BitSet(theMSP->MVT,_MFPRCFLG);
  }
  if (MIVTFct2)
  { theMSP->procID[1]=MIVTFct2;
    BitSet(theMSP->MVT, _MFPRCFLG+1);
  }
  /* Build MIVT (Menu Item Values Table*/
  if (BitTst(&theMF, _MFBLDTAB))
    if (! theMSP->procID[0])
          MSErrExit(progNum,10,menuID,0);
    else MSCallProc (theMSP,menuID,0,0);
  /* Set Menu and TogFlgs for expanded range */
  if (BitTst(&theMF, _MFSETXRG))
    if (!(BitTst(&theMF, _MFEXPRNG)))
        MSErrExit(progNum,06,menuID,0);
    else
    (   defPos=theMV.argT[3];
        if (! defPos)
        { defVal=theMSP->MIVTdefVal;
          defPos=staticCount;
            if (defVal)
              for (i=defPos;i <= dynamCount;i++)
                if (defVal == MSGetMIVT(menuID,i))
                  { defPos=i;break;}
        }
        CheckItem(theMSP->MH,defPos,1);
        MSTogBit(menuID,defPos);
    )
  /* Set Menu and TogFlgs to PREVious state */
  if (BitTst(&theMF, _MFSETPRE))
    if ( ! (theHdl=GetResource(_MSPRVTYP,menuID)) )
        MSErrExit(progNum,12,menuID,0);
    else
    (   HLock(theHdl);
        MSAdjTog(menuID,*theHdl,*theHdl+sizeof(ToggleFlag));
        HUnlock(theHdl);
    )
  /* Execute PROC2 if attached */
  MSCallProc (theMSP,menuID,0,1);
  HUnlock(theMSH);
  return ((int)theMSP->MH);
}

MSInsert(menuID, beforeID)
  short    menuID,beforeID;
( /* Call after MSMake, inserts MST menu in MenuBar*/
  MSHdl        theMSH;
  MenuHandle theMH;
```

```
  short        progNum=04;

  HLock((theMSH=MSGetMSH(menuID,progNum)));
  theMH=(*theMSH)->MH;
  if (! theMH) MSErrExit(progNum,02,menuID,0);
  if(BitTst((*theMSH)->MVT,_MFSUBMEN))
        InsertMenu(theMH,-1);
  else if (beforeID != 0)  InsertMenu(theMH,beforeID);
        else     InsertMenu(theMH,(*theMSH)->beforeID);

  HUnlock(theMSH);
}

MSDispatch(menuID, itemNumber)
  short          menuID;
  short          itemNumber;
{ /*Call after MenuSelect, dispatches to MSDispatch1, which
is recursive, and therefore needs a locked MSPtr */
  MSHdl    theMSH;
  MSPtr    theMSP;
  short    progNum=02;
  int      MIVTFct1;
  Handle theHandle;

  if(!(theMSH=(MSHdl)GetResource(_MSMSTTYP,menuID)) )return;
  HLock(theMSH);
  theMSP=*theMSH;
  MSDispatch1(theMSP,menuID, itemNumber);
  MSCallProc (theMSP,menuID,itemNumber,1);
  HUnlock(theMSH);
}
int MSInsMenuItem(menuID,menuStr,after,theMV)
  short      menuID;
  char     *menuStr;
  short      after;
  MenuVerb *theMV;
{ /* Inserts a dynamic, typed item beyond staticCount, if
     enough space has been allocated*/
  short      progNum=12;
  short      staticCount, dynamicCount,allocCount;
  MSHdl      theMSH;
  MSPtr      theMSP;
  long     menLineTxt=0x02282D00;/* (- pascal string */

  HLock((theMSH=MSGetMSH(menuID,progNum)));
  theMSP=*theMSH;
  MSParseAfter(theMSP,&after,
               &staticCount,&dynamicCount,&allocCount);
  if (dynamicCount >= allocCount)return (-17);
  if (theMV->opCode == _MVTEXTOG
    || theMV->opCode == _MVTEXTOG)
     return (-24);/* not implemented */
  MSShift(theMSP,after,1);
  theMSP->MVT[after+1] = *theMV;
  if (theMV->opCode == _MVMENLIN)
    menuStr=(char *)(&menLineTxt);
  InsMenuItem(theMSP->MH,menuStr,after);
  HUnlock(theMSH);
  return (after+1);
} /* ____________________  End of MSDispatch2
________*/

MSDelMenuItem(menuID,itemNumber)
  short      menuID;
  short      itemNumber;
{ /* Deletes dynamic menu item beyond staticCount*/
  short    progNum=13;
  short    staticCount, dynamicCount,allocCount;
  MSHdl    theMSH;
  MSPtr    theMSP;

  HLock((theMSH=MSGetMSH(menuID,progNum)));
  theMSP=*theMSH;
  MSParseAfter(theMSP,&itemNumber,
               &staticCount,&dynamicCount,&allocCount);
  if (itemNumber == staticCount)
    if (staticCount == dynamicCount) return (-17);
    else itemNumber++;
  MSShift(theMSP,itemNumber,-1);
  DelMenuItem(theMSP->MH,itemNumber);
  HUnlock(theMSH);
  return (0);
}

MSDispose(menuID)
  short    menuID;
{ /* Disposes of menu and associated MST structures*/
  MSHdl      theMSH;
  short      progNum=05;

  theMSH=MSGetMSH(menuID,progNum);
  ReleaseResource((*theMSH)->MH);
  if (BitTst((*theMSH)->MVT, _MFALOCTB))
    DisposHandle((*theMSH)->MIVTH );
  ReleaseResource(theMSH);
}

MSAdjTog(menuID,theTF,theMSK)
  short        menuID;
  ToggleFlag *theTF,*theMSK;
{ /* Sets menus to Toggle Flag snapshot, masked*/
  MSHdl    theMSH;
  MSPtr    theMSP;
  int      i;
  short    allocCount;
  short    progNum=08;

  HLock((theMSH=MSGetMSH(menuID,progNum)));
  theMSP=*theMSH;
  allocCount=(*theMSH)->allocCount;
  for (i=1; i <= allocCount;i++)
    if (BitTst(theMSK,i))
      if (  BitTst(&(*theMSH)->TF,i)
          !=BitTst(theTF,i) )
      { MSDispatch1(theMSP,menuID,i);
        MSCallProc (theMSP,menuID,i,1);
      }
  HUnlock(theMSH);
}

MSAdjPREV(menuID)
  short    menuID;
{ /* Sets stored MST1 resource to current toggle state*/
  MSHdl      theMSH;
  Handle   thePREVH;
  Ptr      thePREVP;
  short      progNum=10;
  int      i;
  short      theErr,allocCount;

  theMSH=MSGetMSH(menuID,progNum);
  if ( ! (thePREVH=GetResource(_MSPRVTYP,menuID))  )
    MSErrExit(progNum,12,menuID,0);
  allocCount=(*theMSH)->allocCount;
  HLock(thePREVH);
  thePREVP=*thePREVH;
  for (i=1; i <= allocCount;i++)
    if (BitTst(thePREVP+sizeof(ToggleFlag),i))
      if (  BitTst(&(*theMSH)->TF,i)
          !=BitTst(thePREVP,i)  )
        if (BitTst(&(*theMSH)->TF,i))BitSet(thePREVP,i);
        else                         BitClr(thePREVP,i);
  ChangedResData(thePREVH);
  if (theErr=ResError())
    MSErrExit(progNum,13,menuID,theErr);
  WriteResource(thePREVH);
  if (theErr=ResError())
    MSErrExit(progNum,13,menuID,theErr);
```

```
  HUnlock(thePREVH);
  ReleaseResource(thePREVH);
}

MSErrExit(progNum,errNum, menuID, itemNumber)
  short progNum,errNum, menuID, itemNumber;
{ /* Puts up an Alert and exits to Finder*/
  Str255   txtStr[3];
  int        i;

  sprintf(txtStr[0],"%d",progNum);
  sprintf(txtStr[1],"%d",errNum);
  sprintf(txtStr[2],"%d",menuID);
  sprintf(txtStr[3],"%d",itemNumber);
  for (i=0;i < 4;i++) CtoPstr(&txtStr[i]);
  ParamText(txtStr[0],txtStr[1],txtStr[2],txtStr[3]);
  StopAlert(500,0);
  ExitToShell();
}

int MSTogBit(menuID,itemNumber)
  short    menuID;
  short    itemNumber;
    /* toggles and returns current value   */
{ /* Toggles flag and returns value */
  MSHdl      theMSH;
  MSPtr      theMSP;
  short      progNum=09;

  theMSH=MSGetMSH(menuID,progNum);
  theMSP=*theMSH;
  if (itemNumber > theMSP->allocCount)
    MSErrExit(progNum,17,menuID,itemNumber);
  if (BitTst(&theMSP->TF,itemNumber))
        { BitClr(&theMSP->TF,itemNumber);return(0); }
  else { BitSet(&theMSP->TF,itemNumber);return(1);   }
}

MSHdl MSGetMSH(menuID,progNum)
  short    menuID;
  short    progNum;
{ /* Gets MST resource handle, on error calls MSErrExit*/
  MSHdl      theMSH;
  if ( (theMSH=(MSHdl)GetResource(_MSMSTTYP,menuID)) )
        return (theMSH );
  else MSErrExit(progNum,01,menuID,0);
}

int MSGetTog(menuID,itemNumber)
  short    menuID;
  short    itemNumber;
{ /* returns current value of toggleflag*/
  MSHdl      theMSH;
  short      progNum=06;

  theMSH=MSGetMSH(menuID,progNum);
  return(BitTst(&(*theMSH)->TF,itemNumber));
}

int MSGetMIVT(menuID,itemNumber)
  short    menuID, itemNumber;
{ /* Retrieves the itemnumber entry from MIVTable*/
  MSHdl      theMSH;
  short      progNum=07;

  theMSH=MSGetMSH(menuID,progNum);
  if (itemNumber > (*theMSH)->allocCount)
    MSErrExit(progNum,17,menuID,itemNumber);
  return (*((*(*theMSH)->MIVTH)+itemNumber));
}


/*_____________  STATIC FCTS  _____________*/
static MSDispatch1(theMSP,menuID, itemNumber)
  MSPtr      theMSP;
  short      menuID;
  short      itemNumber;
{ /* core type-dispatching routine - recursive, therefore must
not lock/unlock MST - calls MSDispatch2 for calculating argT
default values*/
  MenuHandle    theMH;
  MenuVerb      theMV;
  MVOpCode      opCode;
  MVArgT        first, last, parent, other;
  short           progNum=03;
  Str255        theSTR;
  int             theTVal, flag,i;

  if (itemNumber > theMSP->allocCount) return;
  theMH=theMSP->MH;
  theMV=theMSP->MVT[itemNumber];
  opCode=theMV.opCode;
  if (opCode <= _MVNORMAL) return;
  if (itemNumber > theMSP->allocCount)
    MSErrExit(progNum,17,menuID,itemNumber);
  first=theMV.argT[0];
  last=theMV.argT[1];
  parent=theMV.argT[2];
  other=theMV.argT[3];
  theTVal=MSTogBit(menuID,itemNumber);
  MSDispatch2(theMSP,menuID,itemNumber,

opCode,&first,&last,&parent,&other);
  switch (opCode)
  {
      case _MVCHKTOG :
        CheckItem(theMH,itemNumber,theTVal);
      break;
      case _MVTEXTOG :
        if (MSGetIndStr(&theSTR,menuID, other+theTVal) < 0)
              MSErrExit(progNum,08,menuID,itemNumber);
        else SetItem(theMH,itemNumber,&theSTR);
      break;
      case (_MVENABPR) :
        MSTogBit(menuID,other);
        MSEnable(theMH,itemNumber,theTVal);
        MSEnable(theMH,other,! theTVal);
      break;
      case (_MVNZEXRG) :
        MSClear(theMSP,first,last,opCode);
        CheckItem(theMH,itemNumber,1);
        MSTogBit(menuID, itemNumber);
      break;
      case (_MVCMRGRS) :
        switch (itemNumber == parent)
        {
          case(1)  :    /* Parent */
            if (! theTVal)
              MSTogBit(menuID,itemNumber);
            else
            { MSClear(theMSP,first,last,opCode);
              CheckItem(theMH,itemNumber,1);
              if   ((itemNumber >= first)
                &&(itemNumber <= last))
                    MSTogBit(menuID,itemNumber);
            }
          break;
          case (0) :      /* Member */
            CheckItem(theMH,itemNumber,theTVal);
            if (theTVal)
              flag=BitTst(&theMSP->TF,parent);
            else
              flag = ! MSBitOr(theMSP,first,last,opCode);
            if (flag)
            {   CheckItem(theMH,parent,! theTVal);
                MSTogBit(menuID,parent);
            }
          break;
```


© The Definitive MacTutor, Vol. 4

```
        }
      break;
      case (_MVENBSET) :
        if (itemNumber != parent)
          MSTogBit(menuID,itemNumber);
        else
        { for (i=first;i <= last; i++)
            if(theMSP->MVT[i].opCode == opCode)
            {MSEnable(theMH,i,theTVal);MSTogBit(menuID,i);}
          if ((itemNumber >= first) && (itemNumber <= last))
                MSTogBit(menuID,itemNumber);
          else MSEnable(theMH,itemNumber, theTVal);

          itemNumber=other;
          if (MSGetTog(menuID,itemNumber) == theTVal)
            MSDispatch1(theMSP ,menuID,itemNumber)
        }
      break;
      case (_MVTTGSET) :
        if (itemNumber != parent)
          MSTogBit(menuID,itemNumber);
        else
        { for (i=first;i <= last; i++)
            if (theMSP->MVT[i].opCode == opCode)
              if (MSGetIndStr(&theSTR,menuID,
                  theMSP->MVT[i].argT[3]+theTVal) < 0)
                MSErrExit(progNum,08,menuID,i);
              else
              { SetItem(theMH,i,&theSTR);
                MSTogBit(menuID,i);
              }
          if ((itemNumber >= first) && (itemNumber <= last))
            MSTogBit(menuID,itemNumber);
          else
            if (MSGetIndStr(&theSTR,menuID,other+theTVal)<0)
              MSErrExit(progNum,08,menuID,itemNumber);
            else SetItem(theMH,itemNumber,&theSTR);
        }
      break;
  }
}

static MSDispatch2(theMSP,menuID,itemNumber,
                   opCode,first,last,parent,other)
  MSPtr      theMSP;
  short      menuID,itemNumber;
  MVOpCode opCode;
  MVArgT   *first,*last,*parent,*other;
{ /* calculates values for argTs when not specified,i.e.
first&last zero, etc.*/
  int        i,j;
  MenuVerb fakeVerb;
  short      dynamCount;
  short      progNum=11;

  dynamCount=CountMItems(theMSP->MH);
  switch (opCode)
  {
    case (_MVENABPR) :
      if (*other <= 0)
        *other=itemNumber+(*other?-1:1);
    break;
    case (_MVNZEXRG) :
    case (_MVCMRGRS) :
    case (_MVENBSET) :
    case (_MVTTGSET) :
      if (*first && *last) return;
      else if (*first || *last)
              MSErrExit(progNum,14,menuID,itemNumber);
      for (i=itemNumber;i > 0;i--)
        if (theMSP->MVT[i].opCode == _MVMENLIN) break;
      for (j=itemNumber;j <= dynamCount;j++)
        if (theMSP->MVT[j].opCode == _MVMENLIN) break;
      *first=++i;
      *last=--j;
      if (opCode == _MVNZEXRG)break;
      if (*parent <= 0)
        *parent=*first;
      else
      { *parent+=*first-1;
        if (*parent < *first || *parent > *last)
          MSErrExit(progNum,16,menuID,itemNumber);
      }
      if (opCode != _MVENBSET)break;
      if (*other > 0)           break;
      itemNumber=(*other)?*first-2:*last+2;
      if (itemNumber <= 0 || itemNumber > dynamCount)
        MSErrExit(progNum,15,menuID,itemNumber);
      for (i=0;i < _MVARGDIM;i++)
        fakeVerb.argT[i]=theMSP->MVT[itemNumber].argT[i];
      MSDispatch2(theMSP,menuID,itemNumber,_MVCMRGRS,
              &fakeVerb.argT[0],&fakeVerb.argT[1],
              &fakeVerb.argT[2],&fakeVerb.argT[3]);
      *other=fakeVerb.argT[2];
    break;
  }
}

static MSParseAfter(theMSP,after,
                    staticCount,dynamicCount,allocCount)
  MSPtr  theMSP;
  short  *after,*staticCount,*dynamicCount,*allocCount;
{ /*Translates itemNumber defaults for Del/InsMenItem */
  *dynamicCount=CountMItems(theMSP->MH);
  *staticCount=theMSP->staticCount;
  *allocCount=theMSP->allocCount;
  *after+=*staticCount;
  if (*after <= *staticCount)
        *after=*staticCount;
  else if (*after >= *dynamicCount)
        *after=*dynamicCount;
}

static MSShift(theMSP,itemNumber,direction)
  MSPtr    theMSP;
  short    itemNumber, direction;
{ /* Del/InsMenItem utility = shifts MVT by +/- 1 */
  short        progNum=14;
  MenuVerb   *MVTP;
  long       *MIVTP;
  ToggleFlag *TFP;
  int          i,k,first,last,dynamicCount;

  MVTP=theMSP->MVT;
  TFP=&theMSP->TF;
  dynamicCount=CountMItems(theMSP->MH);
  if (BitTst(theMSP->MVT, _MFALOCTB))
        MIVTP=(*theMSP->MIVTH);
  else MIVTP=0;
  if(direction>0) {first=dynamicCount+1;last=itemNumber+1;}
  else            { first=itemNumber;last=dynamicCount;}
  direction*=-1;
  for (i=first;i-last;i+=direction)
  { *(MVTP+i)=*(MVTP+i+direction);
    k=BitTst(TFP,i+direction);
    if (k) BitSet(TFP,i);
    else   BitClr(TFP,i);
    if (MIVTP) *(MIVTP+i)=*(MIVTP+i+direction);
  }
  for(i=0;i < _MVARGDIM;i++) *((char *)(MVTP+last)+i)='\0';
  if (MIVTP) *(MIVTP+last)=theMSP->MIVTnullVal;
  BitClr(TFP,last);
}

static MSCallProc (theMSP,menuID,itemNumber,indx)
  MSPtr    theMSP;
  short    menuID,itemNumber,indx;
```

```
{ /* Gets Handle to and calls MST->procID[indx] PROC parameter
using standard interface (menuID,itemNumber, indx)  - for
MSCallProc itself, indx is the procID entry number */
  int    MIVTFct;
  Handle theHdl;
  int    retVal;
  Str255 theStr;
  short    progNum=16;

    if (! theMSP->procID[indx]) return;
    MIVTFct=theMSP->procID[indx];
    if (! BitTst(theMSP->MVT, _MFPRCFLG+indx))
      if (! (theHdl=GetResource(_MSPRCTYP,MIVTFct)) )
            MSErrExit(progNum,04,menuID,indx+1);
      else {HLock(theHdl);MIVTFct=(int)(*theHdl);}
    retVal=(*((int (*)())(MIVTFct)))
                          (theMSP ,menuID,itemNumber,indx);
    if (retVal < 0)
      MSErrExit(progNum,05,menuID,
                                  (-retVal*256)+indx);
    if (! BitTst(theMSP->MVT,_MFPRCFLG+indx))
      HUnlock(theHdl);
}

static MSEnable(theMH,itemNumber,flag)
  MenuHandle      theMH;
  short           itemNumber,flag;
{ /* Boolean parameter added to ToolBox routine*/
    if (flag)  EnableItem(theMH,itemNumber);
    else     DisableItem(theMH,itemNumber);


}
static MSClear(theMSP,first, last,opCode)
  MenuStuff  *theMSP;
  short      first,last;
  MVOpCode   opCode;
  /*  Clears Toggle Flags and unchecks menu items for
      MenuVerbs in range having same opCode */
{
  short    i,theMark;
  for(i=first;i <= last;i++)
    if(theMSP->MVT[i].opCode == opCode)
    { BitClr(&theMSP->TF,i);
      GetItemMark(theMSP->MH,i,&theMark);
      if (theMark != noMark)
        CheckItem(theMSP->MH,i,0);
    }
}
static MSPcat(s1,s2)
char *s1,*s2;
{int i; BlockMove(s2+1,s1+*s1+1,*s2); *s1+=*s2;}

static int MSBitOr(theMSP,first, last,opCode)
  MSPtr      theMSP;
  short      first,last;
  MVOpCode   opCode;
  /*_______ returns (i..j) togbit k   ________*/
{
  short    i;
  for (i=first;i <= last;i++)
    if (theMSP->MVT[i].opCode == opCode)
      if (BitTst(&theMSP->TF,i))
        return(1);
  return(0);
}

static int MSGetIndStr(theStrP,menuID,strIndx)
  char     *theStrP;
  short      menuID,strIndx;
{ /* Like NOT IN ROM GetIndStr, except returns length,
  or -1 if Indx out of bounds */
  MSHdl      theMSH;
  Handle   theSTRH;
  int        i,maxIndx,len;
  char     *s;
  short      progNum=19;

  theMSH=MSGetMSH(menuID,progNum);
  menuID=(*theMSH)->ttgStrID;
  if( !(theSTRH=GetResource(_MSSTRTYP,menuID))) return (-1);
  maxIndx=*((short *)(*theSTRH));
  if ((strIndx > maxIndx) || (strIndx <= 0)) return (-1);
  s=*theSTRH + sizeof(short);
  for (i=1; i < strIndx; i++) s+=(unsigned char)(*s)+1;

  len=(unsigned char)(*s)+1;
  *theStrP=0;
  BlockMove(s,theStrP,len);
  return(len);
}
```

Listing: MST.C

```
/*  MST.C = minimal MenuStuff Test Program*/
#Options +J +K +Z
#include "MS.h"
#include "Events.h"
#include "Window.h"
#define  _QuitMenNum  2
#define  _QuitItemNum 1
main()
{
  EventRecord  theEvent;
  char         c;
  WindowPtr      theWindow;
  short          windowCode;
  long         menuResult;
  int            i,j;
  Handle       theHandle;
  Str255       theStr;

  InitMenus();
  InitCursor();
  i=CountResources(_MSMSTTYP);
  for (j=1;j <= i;j++)
  { theHandle=GetIndResource(_MSMSTTYP,j);
    GetResInfo(theHandle,&windowCode,&menuResult,&theStr);
    MSMake(windowCode, 0,0);
    MSInsert(windowCode, 0);
  }
  DrawMenuBar();
  FlushEvents(everyEvent);
  while (1)
  { if (GetNextEvent(everyEvent,&theEvent))
      switch (theEvent.what)
      { case (keyDown) :
          c=theEvent.message & 0x000000FFL;
          if (theEvent.modifiers & cmdKey)
            if (c == 'q')
              ExitToShell();
        break;
        case (mouseDown) :
          windowCode=FindWindow(&theEvent.where,&theWindow);
          if (windowCode == inMenuBar)
          {   menuResult=MenuSelect(&theEvent.where);
              MSDispatch(HiWord(menuResult),
                         LoWord(menuResult)  );
              if(HiWord(menuResult) == _QuitMenNum)
                if(LoWord(menuResult) == _QuitItemNum)
                  ExitToShell();
              HiliteMenu(0);
          }
        break;
  } } }
```

Listing: MST.Link

© The Definitive MacTutor, Vol. 4

```
;   MSTst.link
;
/Start QuickStart
/Output MST
MST.Rel
MS.Rel
Standard Library.Rel
/Include MS.rsrc
/Include MSTtst1.rsrc
/Include MSPROC.rsrc
/End
```

Listing: MSProc1.C

```
/*MSPROC1.c = PROC Resource 1,sets MIVT[i] to fontNum */
#Options +J +K +Z
#include "MS.h"
MSPROC1(theMSP,menuID,itemNumber,flg)
  MSPtr         theMSP;
  short         menuID,itemNumber,flg;
{
  short         dynamCount,staticCount,fontNum,i;
  Str255      fontName;
  staticCount=theMSP->staticCount;
  dynamCount=CountMItems(theMSP->MH);
  for (i=staticCount;i <= dynamCount;i++)
  {   GetItem(theMSP->MH,i,&fontName);
      GetFNum(&fontName,&fontNum);
      *((long *)(*theMSP->MIVTH)+i)=fontNum;
  }
  return (0);
}
main()
{}
```

Listing: MSProc1.Link

```
;MSPROC.link
/Output MSPROC1
MSPROC1.Rel
/End
```

Listing: MSProc2.C

```
/* MSPROC2.c =  PROC Resource 2 : when Font menu selected,
sets Font Size to outline if size is RealFont*/
#Options +J +K +Z
#include "MS.h"
#include "Font.h"
MSPROC2(theMSP,menuID,itemNumber,flg)
  MSPtr         theMSP;
  short         menuID;
  short         itemNumber,flg;
{
  int           i;
  short         dynamCount,staticCount,allocCount,
                fontNum,fontSize,byteOffSet,otherMenuID;
  MSHdl         otherMSH;
  long        **MIVTH;
  MenuHandle otherMH;
  Str255      fontName;

  if (! BitTst(theMSP->MVT, _MFPRCPAR))     return(-1);
  if (itemNumber < NULL)
      return(-2);
  dynamCount=CountMItems(theMSP->MH);
  if(itemNumber == NULL) /* may be called at set up */
  { for (i=1;i <= dynamCount;i++)
      if (BitTst(&theMSP->TF,i))
        break;
    itemNumber=(i > dynamCount)?1:i;
  }
  MIVTH=theMSP->MIVTH;
  allocCount=theMSP->allocCount;
  byteOffSet=_MSFIXLEN+(allocCount+1)*sizeof(MenuVerb);
  if ( *(short *)((char *)theMSP+byteOffSet) < 1)
    return(-3); /* must find font menu ID somewhere! */
  fontNum=*(*MIVTH+itemNumber);
  otherMenuID=*((short *)((char *)theMSP+byteOffSet)+1);
  otherMSH=(MSHdl)GetResource(_MSMSTTYP,otherMenuID);
  if (otherMSH == 0) return (-4);
  if (!(otherMH=(*otherMSH)->MH)) return (-4);
  staticCount=(*otherMSH)->staticCount;
  MIVTH=(*otherMSH)->MIVTH;
  dynamCount=CountMItems((*otherMSH)->MH);

  for (i=staticCount;i <= dynamCount;i++)
  { fontSize=*(*MIVTH+i);
    if (RealFont(fontNum,fontSize))
      SetItemStyle(otherMH,i,outlineStyle);
    else
      SetItemStyle(otherMH,i,0);
  }
  return (0);
}
main()
{}
```

Listing: MSProc2.Link

```
;MSPROC2.link - link files for other PROCS similar
/Start QuickStart
/Output MSPROC2
/Strip
MSPROC2.Rel
Standard Library.Rel
/End
```

Listing: MSProc3.C

```
/*MSPROC3.c =  PROC Resource 3, converts Font size text to
numeric in MIVT */
#Options +J +K +Z
#include "MS.h"

MSPROC3(theMSP,menuID,itemNumber,flg)
  MSPtr           theMSP;
  short           menuID;
  short           itemNumber,flg;
{
  int      i,j,k;
  short    dynamCount;
  char   s1[256],s2[256];

  dynamCount=CountMItems(theMSP->MH);

  for (i=1;i <= dynamCount;i++)
  { if (theMSP->MVT[i].opCode == _MVMENLIN)
      continue;
    GetItem(theMSP->MH,i,s1);
    PtoCstr(s1);
    j=0;
    while (! isdigit(*(s1+j)) && j < strlen(s1)) j++;
    if (j < strlen(s1))
    { k=0;
      while (isdigit(*(s1+j)) && j < strlen(s1))
      { *(s2+k)=*(s1+j);
        j++;k++;
      }
      *(s2+k)='\0';
      k=atoi(s2);
      *((*theMSP->MIVTH)+i)=k;
    }
    else *((*theMSP->MIVTH)+i)=0;
  }
  return (0);
}
```

```
main()
()
```

Listing: MSProc3.Link

```
;MSPROC3.link
/Start QuickStart
/Output MSPROC3
MSPROC3.Rel
Standard Library.Rel
/End
```

Listing: MSPROC.R

```
* MSPROC.R = creates PROC
Resources

MSPROC.rsrc

TYPE PROC

  ,1
MSPROC1

  ,2
MSPROC2

  ,3
MSPROC3
```

Listing: MS.R

```
*   Alert resources for
MSErrExit
*

MS.rsrc

Type ALRT

     ,500
66 68 266 427
500
4444

Type DITL

     ,500
10
*   1
BtnItem Enabled
161 262 185 328
Finder

*   2
BtnItem Enabled
159 173 185 244
Restart

*   3
StatText Enabled
11 98 28 304
Menu Stuff Problems !!

*   4
StatText Enabled
54 98 73 342
^0

*   5
StatText Enabled
131 98 152 333
^2

*   6
StatText Enabled
80 98 117 352
^1
```

Listing: MSTtst1.R

```
* Sample Test resources
MSTtst1.rsrc

TYPE MENU

  ,1
\14

  ,2
File
Quit/Q

,3
Test
Font/\1B!\4
Size/\1B!\5
Style/\1B!\6
!\12CheckTog
TextTog 0
EnabPair 1
(EnabPair 2
(-
Enabled Parent 1
Enabled Member 1
(-
(Enabled Parent 2
(Enabled Member 2
(-
TextTog Master
TextTog Member

  ,4
Font

  ,5
Size
9
!\1212
18
20
24

  ,6
Style
!\12Plain
Bold<B
Italic<I
Underlined<U
Outlined<O
Shadowed<S


* File Menu
TYPE MST  = GNRL
,2
* Menu Handle place holder
.H
0000 0000
* Resource type for AddRes-
Menu
.H
0000 0000
* MIVT handle place holder
.H
0000 0000
* procID table
.H
0000 0000 0000 0000
*MIVT default and null values
.H
0000 0000 0000 0000
* static, allocCount,
beforeID, STR# ID
.H
0001 0001 0000 0000
*Toggle Flags
.H
0000 0000 0000
* MVT - zeroeth entry
contains Menu Flag
.H
0000 0000 0000
*StaticCount Menu Verbs
follow :
.H
0000 0000 0000


* Desk Menu
TYPE MST  = GNRL
,1
.H
0000 0000 4452 5652
0000 0000 0000 0000
0000 0000 0000 0000
0000 0000 0000 000F
0002 0000 0000 0000
0000 8000 0000 0000
0000 0000 0000 0000
0000 0000 0000 0000
0000 0000 0000 0000
0000 0000 0000 0000
0000 0000 0000 0000
0000 0000 0000 0000
0000 0000 0000 0000
0000 0000 0000 0000
0000 0000 0000 0000
0000 0000 0000 0000
.H
0000


* Test Menu 3
TYPE MST  = GNRL
,3
.H
0000 0000 0000 0000
0000 0000 0000 0000
0000 0000 0000 0000
0000 0000 0010 0010
0000 0003 0A60 0000
0000 0080 0000 0000
0000 0000 0000 0000
0000 0000 0000 0000
0000 0100 0000 0000
0200 0000 0000 0300
0000 0000 0300 0000
FF00 8000 0000 0000
0600 0000 0000 0600
0000 0000 8000 0000
0000 0600 0000 FF00
0600 0000 FF00 8000
0000 0000 0700 0000
0000 0700 0000 0000


* Size Menu
TYPE MST  = GNRL
,5
.H
0000 0000 0000 0000
0000 0000 0000 0003
0000 0000 0000 0000
FFFF FFFF 0001 0005
0000 0000 0000 0000
0000 3D00 0000 0000
0400 0000 0200 0000
0000 0000 0000 0000
0000 0000 0000 0000
.H
0000 0000 0000

* Font Menu
TYPE MST  = GNRL
,4
.H
0000 0000 464F 4E54
0000 0000 0000 0001
0000 0002 0000 0000
FFFF FFFF 0001 0006
0000 0000 0000 0000
0000 FD40 0000 0000
0400 0000 0100 0000
0000 0000 0000 0000
0000 0000 0000 0000
0000 0000 0000 0000
.H
 0000 0000
*****   PROC PARAM
.H
0001 0005


* STYLE Menu
TYPE MST  = GNRL
,6
.H
0000 0000 0000 0000
0000 0000 0000 0000
0000 0000 0000 0000
0000 0000 0006 0006
0000 0000 4000 0000
0000 0100 0000 0000
0500 0000 0000 0500
0000 0000 0500 0000
0000 0500 0000 0000
0500 0000 0000 0500
.H
0000 0000

TYPE STR#

* Text Toggled strings
     ,3 (32)
6
TextTog 1 - 0
TextTog 1 - 1
TextTog Master - 0
TextTog Master - 1
TextTog Member - 0
TextTog Member - 1
```

© The Definitive MacTutor, Vol. 4

Programmer's Workshop

HFS Transfer DA

Clifford Story
Jacksonville, FL
MacTutor Vol. 4 No. 10

Transfer DA

The Transfer DA is a small desk accessory that adds a Transfer menu to any application that supports desk accessories.

When Apple killed the Minifinder, my work habits were shattered. I tried Oasis, a Finder substitute, for a while but I didn't like the way the working directory was changed by launching a program; that is, if I launched, say, Macwrite, and opened a file, the sfget dialog came up showing the directory that contained Macwrite, rather than the one I had been working in. I had to navigate through folders to return to my work files.

Transfer solves that problem. Now, I work on the desktop but when I want to switch from one application to another, I use Transfer. The working directory is unaffected, and I skip the Finder.

This DA illustrates some interesting programming techniques: it is a segmented DA; it does an HFS search of an entire hard disk; it uses the iaznotify hook; and it creates resources with TML Pascal (a subject which came up in the May Mac Tutor letters column).

How the DA Works

When the DA is launched for the first time, it starts at the root of the disk which contains the System folder, and does an HFS catalog of the entire disk. It finds each application on the disk, and stores the name and directory id in a special resource FILE called "Transfer data", in the System folder. Henceforth, it uses this file to build the menu (the HFS catalog takes a little time - about 30 seconds on my 20 meg hard disk - so you wouldn't want to go through it every time).

When an application is selected from the Transfer menu, the applications's name and directory id are copied into the DA's global data record, and the address of a special launch routine is installed in the "iaznotify" low-memory global. The transfer may be immediate or delayed; in the first case, the DA will call "ExitToShell"; in the second, it will wait for the application to quit normally.

When the application quits, the Mac will prepare to launch the next application (usually the Finder). As one of the last steps in this process, it will call "InitApplZone" to initialize the heap. This trap will, in turn, call the routine whose address is stored in "iaznotify" - the DA's launch routine. This routine simply writes the name of the desired application to "curappname" (another low-memory global), and sets the directory to the proper directory. The Mac will then go ahead and launch the application. Sort of a cowbird launch, you might say!

DA Segmentation

This is a small DA, a bit over 6K. so why is it segmented? Am I just showing off?

Before "InitApplZone" calls the launch procedure, the Mac closes all resource files. That means that if the launch procedure is in a resource (e.g., the DRVR resource), it will be left in a free block. I don't feel very secure when executing code in a free block! The solution is to detach the resource; the block will remain allocated until "InitApplZone" actually cleans up the heap - which happens after it calls the launch routine.

But if the launch routine is part of the DRVR resource, this means detaching the DRVR. If we do that, we'll never know about menu selections, since the desk Manager needs a DRVR resource to pass DA events to. So the launch routine has to be in a separate segment.

The launch routine is in a PACK resource. The DRVR's "open" routine loads and locks the PACK, and records its address. When it wants to call a routine from the PACK, it puts a routine selector into register D0 and jumps to the PACK. At the top of the PACK is a jump table that branches to the routine corresponding to the selector.

There are other ways to segment a DA; this is just the way I did this one.

Creating Resources with TML Pascal

Since it came up in the May letters column, let me say a few words on this topic. It's really simple, once you know the (undocumented) trick. It's all in the link control file.

For example, here is a link control file to create an LDEF:

```
!fastlist
/codetype LDEF 1001 'Fast List' 32
list
pas$library
macintfglue
/end
```

The first line is the linker's start point - the name of the main routine. The second line defines the resource type. This will be an LDEF, resource id 1001, named "Fast List", purgeable. The next three lines are the object files to link, in the order you want them to be linked. And the final line means just what it says.

The code itself should be compiled as a unit, and most resource types will require that the main routine appear at the very top of the resource.

The two resources in this program are linked a bit differently, the DRVR resource because the resource definition requires a DA header at the top, and the PACK resource because I want a jump table at the top. The fact that I left out the first line and got away with it suggests that it may not be necessary.

Using the DA

I have tested this DA with every application on my hard disk (at the time I wrote it). A number of applications require the delayed transfer; for example, any program that uses scratch files. I found only one that would not work at all. Can you guess? That's right - Microsoft Word.

Transfer vs. Multifinder

I leave as an exercise for the reader the problem of making this DA work with the Multifinder. A more fruitful project might be converting the DA to an FKEY.

```
;****************************************
;
;    types.asm
;    ——————
;
;    (c) 1987, 1988 Attic Software
;
;    record formats for Transfer
;
;****************************************

;****************************************
;    wrecord
;****************************************

    WRthemenu     equ        0; MenuHandle
    WRresfactor   equ        4; integer
    WRsysdir      equ        6; long
    WRpackaddr    equ       10; Ptr
    WRlaunchpath  equ       14; long
    WRlaunchaddr  equ       18; long
    WRiazaddr     equ       22; long
    WRtheevent    equ       26
                  ; EventRecord
    WRthename     equ       42; Str255
    WRtheblock    equ      298; hfs block

;****************************************
```

```
(****************************************
types.pas
——————
(c) 1987, 1988 Attic Software

Pascal declarations for Transfer

****************************************)

unit types;

(****************************************)

interface

(****************************************)

uses macintf, hfs;

(****************************************)

type

  logical          =   boolean;
  long             =   longint;

  shortpointer     =   ^integer;
  longpointer      =   ^long;

(****************************************

Program constants consist of (1) DA message types (most of
which are not used in Transfer); (2) low-memory globals; and
(3) resource ids.

****************************************)

const

  accEvent         =   64;
  accRun           =   65;
  accCursor        =   66;
  accMenu          =   67;
  accUndo          =   68;
  accCut           =   70;
  accCopy          =   71;
  accPaste         =   72;
  accClear         =   73;
  goodbyekiss      =   -1;

  curappname       =   $910;
  erik             =   $4552494B;
  findername       =   $2E0;
  iaznotify        =   $33C;
  sysmap           =   $A58;

  menunum          =   1001;
      aboutitem    =   1;
      builditem    =   2;
      finderitem   =   4;

  errordialog      =   1001;
  aboutdialog      =   1002;
  builddialog      =   1003;
  delaydialog      =   1004;

  packnum          =   1001;
  stringnum        =   1001;

  datastart        =   1000;

(****************************************
A "deskrecord" is simply an application's name and directory.
We need to know both of these to transfer to that application.
An "marray" is an array of deskrecords, along with the array's
size.
Finally, a "wrecord" is the DA's  global data.  The last three
```

```
fields ("theEvent", "thename" and the varient parameter block)
are just scratch areas, used by various routines as local
variables.  Putting them here keeps them off the stack.

*************************************)

type

  deskrecord        =   record
      dirid             :   long;
      name              :   Str255;
  end;
  deskpointer       =   ^deskrecord;
  deskhandle        =   ^deskpointer;

  marray            =   record
      count             :   integer;
      data              :   array
                        [0..1000] of deskhandle;
  end;
  arraypointer      =   ^marray;
  arrayhandle       =   ^arraypointer;

  wrecord           =   packed record
  theMenu           :   MenuHandle;         resfactor
  :   integer;          sysdir              :       long;
      packaddr          :   Ptr;
  launchpath        :   long;               launchaddr
  :   long;
  iazaddr           :   long;
  theEvent          :   EventRecord;
  thename           :   Str255;  case integer of
          1   :   (infoblock
                        :   CInfoPBRec);
                  2   :   (hblock
                        :   HParamBlockRec);
                  3   :   (wdblock
                        :   WDPBRec);
                  end;
  wpointer          =   ^wrecord;

(*************************************)

implementation

(*************************************)

end.
```

```
(*************************************)

;*************************************;
; drvrhead.asm
; ———————
;
; (c) 1987, 1988 Attic Software
;
; header for DRVR segment of
; Transfer.
;
;*************************************

;*************************************
; includes
;*************************************

  include  sysequ.d
  include  types.asm

;*************************************
; imported routines
;*************************************

  xref     open
  xref ctl

;*************************************
; exported routines
;*************************************

  xdef     initglobals
  xdef     setdir

;*************************************
;
; The canonical DA header.  This is a
; modified version of the DA header
; supplied with TML Pascal.
;
;*************************************

  dc.w     $0400           ; flags
  dc.w     $0000           ; service        dc.w     $ffff
; mask
  dc.w     $0000           ; menuID dc.w    ornopen  ; open
dc.w       orndone         ; prime
  dc.w     ornctl          ; Control
  dc.w     orndone         ; Status
  dc.w     ornclose        ; close

;*************************************

  dc.w     '(c) 1987, 1988 '
  dc.w     'Attic Software'
  dc.w     'All rights reserved'

;*************************************

ornopen
  movem.l  A0/A1,-(SP)
  move.l   A1,-(SP)        ; device
  move.l   A0,-(SP)        ; param  jsr     open
  movem.l  (SP)+,A0/A1
  move.l   jiodone,-(SP)
  rts

ornctl
  movem.l  A0/A1,-(SP)
  move.l   A1,-(SP)        ; device
  move.l   A0,-(SP)        ; param  jsr     ctl
  movem.l  (SP)+,A0/A1
  move.l   jiodone,-(SP)
  rts

ornclose
orndone
  move.l   jiodone,-(SP)
  rts

;*************************************
;
; The routine loader moves a selector
; into D0, and the address of the
; PACK segment into A0.  It then
; jumps to the pack (the pack begins
; with a jump table).
;
;*************************************

initglobals
  move.w   #0,D0
  bra.w    loader

setdir
  move.w   #4,D0
  bra.w    loader

loader
```

```
  movea.l  4(SP),A0        ; globals
  movea.l  WRpackaddr(A0),A0
  jmp      (A0)

;****************************************
```

```
(****************************************

drvr.pas
```

```
(c) 1987, 1988 Attic Software

Pascal routines for DRVR segment of Transfer

****************************************)

unit drvr;

(****************************************)

interface

(****************************************)

uses macintf, hfs, types;

(****************************************)

implementation

(****************************************)

procedure initglobals
  (globals : wpointer); external;
function setdir
  (dirid : long;
  globals : wpointer) : OSErr; external;

(****************************************

drawscreen
```

```
This routine draws the "About" message in its rectangle.

****************************************)

procedure drawscreen(
  theWindow : WindowPtr;
  theitem : integer);

var
  theType          :   integer;
  theHandle        :   Handle;
  thebox           :   Rect;

begin

  SetPort(theWindow);

  GetFNum('monaco', theType);
  TextFont(theType);
  TextSize(9);

  GetDItem(theWindow, theitem, theType,
      theHandle, thebox);

  theHandle := GetResource('INFO',
      GetWRefCon(theWindow));
  HLock(theHandle);
  TextBox(theHandle^,   GetHandleSize(theHandle),
      thebox, 0);
  HUnlock(theHandle);

end;

(****************************************

transalert
```

I want all the alerts to identify their source, so I made them dialogs, in titled windows, and wrote this routine to imitate "Alert".

This is not the best way to do it; for one thing, a titled window should be dragable. Modal dialogs, therefore, should not have titles. I should have simply added a distinctive icon to the item lists.

Since this is the first use of the "resfactor" field of the global record, it's as good a place as any to explain it. This DA is given a formal resource id of 16, and all its owned resources are numbered accordingly. The Font/DA Mover will change all these numbers when it installs the DA. "resfactor", which is computed in the "open" routine, below, is the correction factor that converts the hard-coded resource ids to the actual ids in use.

Note that "resfactor" doesn't convert the formal ids; it actually converts the constants defined in the "types.pas" unit, which are nice positive numbers, equal to the resource ids plus 16872.

There's a little bit of code to install the previous procedure if this is the "About" dialog.

```
****************************************)

procedure transalert
  (dialognum : integer;
  globals : wpointer);

var
  savedport          :   GrafPtr;
  theDialog          :   DialogPtr;
  therecord          :   DialogRecord;
  theType        :   integer;
  theHandle          :   Handle;
  thebox         :   Rect;
  choice         :   integer;

begin

  with globals^ do begin

      GetPort(savedport);

      theDialog := GetNewDialog(dialognum
          + resfactor, @therecord,
          pointer(-1));
      SetPort(theDialog);

      if dialognum = aboutdialog then
                                    begin
          GetDItem(theDialog, 2, theType,
              theHandle, thebox);
          SetDItem(theDialog, 2, theType,
              Handle(@drawscreen), thebox);
          SetWRefCon(theDialog, dialognum
              + resfactor);
      end;

      InitCursor;
      ShowWindow(theDialog);
```


© The Definitive MacTutor, Vol. 4

```
      repeat
          ModalDialog(nil, choice);
      until choice = ok;

      CloseDialog(theDialog);
      SetPort(savedport);

  end;

end;

(*****************************************

errordisplay
————

  This routine displays error messages.  The texts of the
messages are in a string list.

*****************************************)

procedure errordisplay
  (appnum, sysnum, resnum : long;
  globals : wpointer);

var
  string1              :   Str255;
  string3              :   Str255;
  string4              :   Str255;

begin

  InitCursor;

  with globals^ do begin

      GetIndString(string1, resfactor
          + stringnum, resnum);
      if string1 = '' then
          string1  := 'An error has occurred!';

      NumToString(appnum, string3);
      NumToString(sysnum, string4);
      ParamText(string1, '',
          string3, string4);

      SysBeep(10);
      transalert(errordialog, globals);

  end;

end;

(*****************************************

getapps
——

  Here it is - the only recursive routine I have ever been
forced to use!  (Recursion is overrated; it is, in my opinion,
better to avoid recursion, if you can do it in a natural
fashion.  Loops are easier to read, and generally more
efficient.)

  This routine is passed a directory id, the count of objects
(files and folders) in that directory, and the volume refer-
ence number.  It indexes through the directory with "PBGet-
CatInfo".

  If the object is another directory, it calls itself with
that directory's id and count.

  If the object is a file, and if the file is an application,
it records its name and the directory id in the application
array.

*****************************************)

procedure getapps
  (thedir : long; thecount : integer;
  thevol : integer;
  theHandle : arrayhandle;
  globals : wpointer);

label
  100;

var
  index              :   integer;
  anerror            :   integer;
  theDialog              :   DialogPtr;
  jndex              :   integer;
  thelength              :   integer;
  thedesk            :   deskhandle;

begin

  with globals^ do begin

      for index := 1 to thecount do
          with infoblock do begin

          thename := '';

          ioCompletion := nil;
          ioNamePtr := @thename;
          ioVRefNum := thevol;
          ioFDirIndex := index;
          ioDrDirID := thedir;

          anerror:= PBGetCatInfo(@infoblock,
                  false);
          if anerror <> noErr then begin
              errordisplay(101, anerror,
                  2, globals);
              goto 100;
          end;

          if BitAnd(ioFlAttrib, $10) = $10 then
              getapps(ioDrDirID, ioDrNmFls,
                  thevol, theHandle, globals)
          else if ioFlFndrInfo.fdType
                          = 'APPL' then begin

              jndex := theHandle^^.count+ 1;
              theHandle^^.count := jndex;

  SetHandleSize
                  (Handle(theHandle),
                  5 * (jndex + 1));

              HLock(Handle(theHandle));

              with theHandle^^ do
                  while (IUCompString(thename,
                  data[jndex - 1]^^.name) < 0)
                  and (jndex > 1) do begin

                  data[jndex]:= data[jndex - 1];
                  jndex := jndex - 1;

              end;

              thelength:= 10 + length(thename);
              thelength:= 2 * (thelength div 2);
              theHandle^^.data[jndex]
                  := deskhandle
                      (NewHandle(thelength));
```

```
            with theHandle^^.data[jndex]^^
                        do begin
                dirid := thedir;
                name := thename;
            end;

            HUnlock(Handle(theHandle));

        end;

100:  end;

  end;

end;

(****************************************

walktree
--------

  This routine catalogs all the applications on a given disk.

  It first puts up a dialog, telling the user what's going on.

  It next calls "PBGetCatInfo" for the root directory (directory id = 2), to get the number of objects in the root.  Then it calls "getapps" to walk the HFS tree recursively.

  The collected data is written to the current resource file ("Transfer Data", in the System folder), and the dialog is dismissed.

****************************************)

procedure walktree
  (thevol : integer;
  theHandle : arrayhandle;
  globals : wpointer);

var
  savedport             :   GrafPtr;
  theDialog             :   DialogPtr;
  therecord             :   DialogRecord;
  index             :   integer;
  anerror           :   integer;

begin

  with globals^ do begin

      GetPort(savedport);

      theDialog := GetNewDialog(resfactor
          + builddialog, @therecord,
          pointer(-1));
      SetPort(theDialog);
      ShowWindow(theDialog);
      DrawDialog(theDialog);

      with infoblock do begin
          ioCompletion := nil;
          ioNamePtr := nil;
          ioVRefNum := thevol;
          ioFDirIndex := 0;
          ioDrDirID := 2;
      end;
      anerror := PBGetCatInfo(@infoblock,
          false);
      if anerror <> noErr then
          errordisplay(102, anerror,
              2, globals)
          else
              getapps
                  (2, infoblock.ioDrNmFls,
                  thevol, theHandle, globals);
          end;

  HLock(Handle(theHandle));
  with theHandle^^ do
      for index := 1 to count do begin

          AddResource
              (Handle(data[index]), '.Trn',
              datastart + index,
              data[index]^^.name);

          SetHandleSize
              (Handle(data[index]), 4);

      end;
  HUnlock(Handle(theHandle));

  CloseDialog(theDialog);
  SetPort(savedport);

end;

(****************************************

buildmenu
---------

  This routine assembles the necessary data, and builds the Transfer menu.

  It finds the volume reference number of the disk with the System folder, sets the directory to the System folder, and opens or creates the "Transfer Data" file in that directory.

  If this file lacks a header resource (whether because it was just created, or because it has been corrupted), then it must be rebuilt, with "walktree".

  Then the menu is built.  The menu resource is loaded, and the fourth item set to the name of the current Finder.  The remainder of the menu is copied from the resource file.

****************************************)

procedure buildmenu(globals : wpointer);

label
  100;

var
  thepointer            :   shortpointer;
  thevolume                 :   integer;
  theres                :   integer;
  theHandle                 :   arrayhandle;
  index                 :   integer;
  jndex                 :   integer;
  thedesk               :   Handle;
  theID                 :   integer;
  theType               :   ResType;
  anerror               :   integer;

begin

  with globals^ do begin

      thepointer := shortpointer(sysmap);
      anerror := GetVRefNum
          (thepointer^, thevolume);
      if anerror <> noErr then begin
          errordisplay(103, anerror,
              3, globals);
          goto 100;
```


© The Definitive MacTutor, Vol. 4

```
    end;

    anerror := setdir(sysdir, globals);
    if anerror <> noErr then begin
        errordisplay(104, anerror,
            3, globals);
        goto 100;
    end;

    thename := 'Transfer Data';
    theres := OpenResFile(thename);
    if ResError = fnfErr then begin
        CreateResFile(thename);
        theres := OpenResFile(thename);
    end;
    if ResError <> noErr then begin
        errordisplay(105, ResError,
            3, globals);
        goto 100;
    end;

    theHandle
        := arrayhandle
            (Get1Resource('.Trn',
            datastart));
    if theHandle = nil then begin
        theHandle := arrayhandle
            (NewHandle(6));
        theHandle^^.count := 0;
        theHandle^^.data[0]
            := deskhandle(NewHandle(16));
        AddResource(Handle(theHandle),
            '.Trn', datastart, '');
        walktree(thevolume,
            theHandle, globals);
        SetHandleSize
            (Handle(theHandle), 2);
    end;

    theMenu := GetMenu(resfactor
        + menunum);
    BlockMove(Ptr(findername),
        Ptr(@thename), 16);
    SetItem(theMenu, 4, thename);

    jndex := 1;
    for index := 1 to theHandle^^.count
                                do begin
        thedesk := Get1Resource('.Trn',
            datastart + index);
        if thedesk <> nil then begin
            AppendMenu(theMenu, '.Trn');
            GetResInfo(thedesk, theID,
                theType, thename);
            SetItem(theMenu, jndex + 4,
                thename);
            jndex := jndex + 1;
        end;
    end;

    InsertMenu(theMenu, 0);
    DrawMenuBar;

    CloseResFile(theres);

100:

  end;

end;

(*************************************

systemvol
```

This routine is more or less straight out of Tech Note 77, pages 3 and 4. It returns a working directory reference number for the System folder, suitable for use in file system calls.

Step one is to find the volume reference number of the volume that holds the System folder. "sysmap" is the file reference number of the System file (an open file), so "GetVRefNum" will find the volume refence number of the System file and, of course, the System folder.

Step two is to get the directory id, with a call to "PBHGetVInfo". The directory id is returned in the "ioVFndrInfo[1]" field of the HParamBlockRec.

Finally, "PBOpenWD" will return the System folder's working directory reference number, which can be used as a volume reference number in file system calls.

```
*************************************)

function systemvol
  (globals : wpointer) : integer;

var
  thepointer            :   shortpointer;
  thevolume             :    integer;
  anerror               :   integer;

begin

  with globals^ do begin

      thepointer := shortpointer(sysmap);
      anerror := GetVRefNum(thepointer^,
          thevolume);

      with hblock do begin
          ioNamePtr := nil;
          ioVRefNum := thevolume;
          ioVolIndex := 0;
      end;
      anerror := PBHGetVInfo(@hblock,
          false);

      with wdblock do begin
          ioWDDirID
              := hblock.ioVFndrInfo[1];
          ioNamePtr := nil;
          ioVRefNum := thevolume;
          ioWDProcID := erik;
      end;
      anerror := PBOpenWD(@wdblock,
          false);

      systemvol := wdblock.ioVRefNum;

  end;

end;

(*************************************

rebuildmenu
```

If the "Rebuild menu" item is chosen from the menu, or an application is chosen which can't be found, Transfer will rebuild the menu from scratch. It does this by deleting the "Transfer Data" file, and calling "buildmenu".

```
*************************************)
```

```
procedure rebuildmenu
  (globals : wpointer);

var
  anerror           :    integer;

begin

  with globals^ do begin

       DeleteMenu(resfactor + menunum);
       ReleaseResource(Handle(theMenu));
       anerror := FSDelete
           ('Transfer Data',
           systemvol(globals));
       buildmenu(globals);

  end;

end;

(****************************************

dofinder
--------

  If the "Finder" item is chosen from the menu, then no
transfer is desired, so restore the "iaznotify" hook to the
value it held when Transfer was launched. (This isn't quite
right, since something besides Transfer may have changed it
since then.  but I don't see any way to correct for that...)

  If the option key is down, do nothing else.  Otherwise, do
an immediate transfer by calling "ExitToShell".

****************************************)

procedure dofinder(globals : wpointer);

var
  thepointer        :    longpointer;

begin

  with globals^ do begin

       thepointer
           := longpointer(iaznotify);
       thepointer^ := iazaddr;

       if GetNextEvent(0, theEvent) then
           ;
       if BitAnd(theEvent.modifiers,
                       optionKey) = 0 then
           ExitToShell;

  end;

end;

(****************************************

clickmenu
---------

  The first few menu choices are handled by routines above.

  If an application is chosen, we need to get (1) the
application's name, and (2) its directory.  The name is easy;
it's on the menu.  To get the directory, we have to go back to
the "Transfer Data" file.

  Once we have the application's directory, the next thing to
do is to make sure it's there.  Transferring to a non-existent
application will cause a system bomb.

  If everything is ok, then if the option key is down, prepare
to do a delayed transfer; otherwise, do an immediate transfer
by calling "ExitToShell".

****************************************)

procedure clickmenu
  (itemchoice : integer;
  globals : wpointer);

label
  100;

var
  theres                :    integer;
  thedesk               :    deskhandle;
  theinfo               :    FInfo;
  thepointer            :    longpointer;
  anerror               :    integer;

begin

  with globals^ do begin

       case itemchoice of
           aboutitem     :
               transalert(aboutdialog,
                   globals);
           builditem     :
               rebuildmenu(globals);
           finderitem    :
               dofinder(globals);
       otherwise

           thepointer := longpointer
               (iaznotify);
           thepointer^ := launchaddr;

           anerror := setdir(sysdir,
               globals);
           if anerror <> noErr then begin
               errordisplay(106, anerror,
                   3, globals);
               goto 100;
           end;

           theres := OpenResFile
               ('Transfer Data');
           if ResError <> noErr then begin
               rebuildmenu(globals);
               goto 100;
           end;

           GetItem(theMenu,
               itemchoice, thename);
           thedesk := deskhandle
               (Get1NamedResource('.Trn',
               thename));
           DetachResource(Handle(thedesk));
           CloseResFile(theres);

           if thedesk = nil then begin
               rebuildmenu(globals);
               goto 100;
           end;

           anerror := setdir
               (thedesk^^.dirid, globals);
           if anerror <> noErr then begin
               rebuildmenu(globals);
               goto 100;
```


© The Definitive MacTutor, Vol. 4

```
        end;

        anerror := GetFInfo(thename,
            0, theinfo);
        if anerror <> noErr then begin
            rebuildmenu(globals);
            goto 100;
        end;

        launchpath := thedesk^^.dirid;

        if GetNextEvent(0, theEvent) then
            ;
        if BitAnd(theEvent.modifiers,
                        optionKey) = 0 then
            ExitToShell
        else begin
            ParamText(thename,
                '', '', '');
            transalert(delaydialog,
                globals);
        end;

    end;

100:  HiliteMenu(0);

  end;

end;

(****************************************

open
—

  This is the canonical DA open routine.  If the DA has
already been opened, device.dCtlMenu will be nonzero; do
nothing.  Otherwise, Allocate the globals, and fill in a few
fields.  Of particular interest is the calculation of "resfac-
tor" by the magic formula $BFE0 - 32 * dCtlRefNum - 1000.
This, assuming I have given the DA the formal resource id of
16, allows be to refer to owned resources by ids from 1000 to
1031, adding resfactor to convert to the actual values.

  Next, load and detach the PACK segment.  And lock it; it's
created locked but why take chances?

  Finally, call "initglobals" to fill in the rest of the
fields, and "buildmenu" to set up the menu.

****************************************)

procedure open(var device : DCtlEntry;
  var block : ParamBlockRec);

var
  globals               :   wpointer;
  packhandle            :   Handle;

begin

  if device.dCtlMenu = 0 then begin

      globals := wpointer
          (NewPtr(sizeof(wrecord)));
      if globals <> nil then
          with globals^ do begin

          with device do begin
              resfactor := $BFE0
                  - 32 * dCtlRefNum - 1000;
              dCtlMenu := resfactor
                  + menunum;
              dctlwindow := nil;
              dCtlStorage
                  := Handle(globals);
          end;

          packhandle := GetResource
              ('PACK', resfactor + packnum);
          DetachResource(packhandle);
          HLock(packhandle);
          packaddr := packhandle^;

          initglobals(globals);
          buildmenu(globals);

      end;

  end;

end;

(****************************************

ctl
—

  The canonical Control routine.  The only events we're
interested in are menu clicks; if we get one, call "click-
menu".

****************************************)

procedure ctl(var device : DCtlEntry;
  var block : ParamBlockRec);

var
  globals               :   wpointer;

begin

  if (device.dCtlMenu <> 0)
      and (block.csCode = accMenu)
                              then begin
      globals := wpointer
          (device.dCtlStorage);
      clickmenu
          (block.csParam[1], globals);
  end;

end;

(****************************************)

  end.

(****************************************)
```

```
/codetype DRVR 16 'Transfer HD' 16
drvrhead
drvr
pas$library
macintfglue
/end
```

```
;****************************************
;
;     packhead.asm
;     ————
;
;     (c) 1987, 1988 Attic Software
;
;     header and assembly routines for
;     PACK segment of Transfer.
```

```
;
;****************************************

;****************************************
;      exported routines
;****************************************

      xdef   setglobal
      xdef   getglobal
      xdef   runiaz

;****************************************
;
;      The jump table begins with a jump
;      into the body of the table, indexed
;      by D0 (the routine selector).  Then
;      it branches to the appropriate
;      routine.
;
;****************************************

      xref   initglobals
      xref   setdir

      jmp    2(PC,D0.w)
      bra.w  initglobals
      bra.w  setdir

;****************************************
;
;      The global access routines provide
;      a way to save the address of the
;      global record, which would
;      otherwise be lost.  This is done by
;      writing the address into a four-
;      byte constant in the PACK segment.
;
;      Note:  writing to code segments is
;      frowned upon in some circles.  This
;      objection has a legitimate basis:
;      self-modifying code is a horror.
;      It makes debugging difficult, and
;      maintenance impossible.  (5 points:
;      where in "Inside Mac" are you
;      instructed to write self-modifying
;      code?)  In this instance, I an not
;      modifying code; I am modifying
;      data, and there's nothing wrong
;      with that.
;
;      procedure setglobal(value : long);
;      function getglobal : long;
;
;****************************************

      module 'access'

setglobal
      movea.l             (SP)+,A0
      lea    dummy,A1
      move.l (SP)+,(A1)
      jmp    (A0)

getglobal
      movea.l             (SP)+,A0
      move.l dummy,(SP)
      jmp    (A0)

dummy
      dc.w   'xxxx'

;****************************************
;
;      procedure runiaz(iazaddr : long)
;
;      The runiaz routine zeros the
;      "iaznotify" hook, and calls the
;      routine that preceded Transfer's
;      routine, if any.
;
;****************************************

      module 'runiaz'

runiaz
      clr.l  $33C
      movea.l             4(SP),A0
      move.l (SP)+,(SP)

      cmpa.l #0,A0
      beq.w  L0001
      move.l A0,-(SP)

L0001
      rts

;****************************************
```

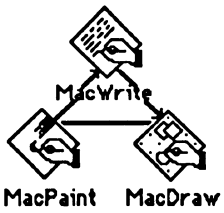
```
(****************************************

pack.pas
----

(c) 1987, 1988 Attic Software

Pascal routines for PACK segment of Transfer

****************************************)

unit pack;

(****************************************)

interface

(****************************************)

uses macintf, hfs, types;

(****************************************)
implementation
(****************************************)

procedure setglobal
    (value : long); external;
function getglobal : long; external;
procedure runiaz
    (iazaddr : long); external;

(****************************************
systemdir
----
    This routine is more or less straight out of Tech Note 67.
It returns a directory id for the System folder, suitable for
use in "SetVol" calls.
    Step one is to find the volume reference number of the
volume that holds the System folder.  "sysmap" is the file
reference number of the System file (an open file), so
"GetVRefNum" will find the volume refence number of the System
file and, of course, the System folder.  (The Tech Note skips
this step; it searches the boot drive for a System folder, and
may not find one.)
    Step two is to get the directory id, with a call to
"PBHGetVInfo".  The directory id is returned in the
"ioVFndrInfo[1]" field of the HParamBlockRec.
****************************************)
function systemdir
```

```
    (globals : wpointer) : long;
var
    thepointer          :   shortpointer;
    thevolume           :   integer;
    anerror             :   integer;
begin
    with globals^ do begin
        thepointer := shortpointer(sysmap);
        anerror := GetVRefNum(thepointer^,
            thevolume);
        with hblock do begin
            ioNamePtr := nil;
            ioVRefNum := thevolume;
            ioVolIndex := 0;
        end;
        anerror := PBHGetVInfo(@hblock,
            false);
        systemdir := hblock.ioVFndrInfo[1];
    end;
end;

(****************************************
setdir
---
    This is just a shell around "PBHSetVol".
****************************************)
function setdir
    (dirid : long;
    globals : wpointer) : OSErr;

begin

    with globals^ do begin

        with wdblock do begin
            ioCompletion := nil;
            ioNamePtr := nil;
            ioVRefNum := 0;
            ioWDDirID := dirid;
        end;

        setdir := PBHSetVol(@wdblock,
            false);

    end;
end;

(****************************************
postlaunch
---
    This is the routine that performs the actual launch.  It
is called by "InitApplZone" throught the "iaznotify" hook.
    First, recover the global record with a call to "getglo-
bal".  This routine is called after all the resources have
been released; the DRVR segment is no longer around, and even
if it was, "postlaunch" isn't called by it, so we can't find
the globals in the usual way.  "getglobal" will return a long
word stored right in the PACK segment, which has been previ-
ously set to a pointer to the globals.
    Next, "postlaunch" will call any routines that were
already installed in "iaznotify" when "postlaunch" was
installed there, via the "runiaz" routine.
    Then it calls "setdir" to set the volume to the folder
containing the chosen application, and copies the
applications's name to "curappname".  The system will now be
fooled into launching that application instead of the Finder.
****************************************)
procedure postlaunch;

var
    globals             :   wpointer;
    anerror             :   integer;

begin

    globals := wpointer(getglobal);
    with globals^ do begin
        runiaz(iazaddr);
        anerror := setdir(launchpath,                glo-
bals);
        if anerror = noErr then
            BlockMove(@thename,
Ptr(curappname), 32);
    end;

end;
(****************************************
initglobals
---
    Initialize global data.  Note that we preserve the value
at "iaznotify"; if the current application has installed a
routine there, we will want to run it before we run ours.
    Also, we don't want to confuse the new application with
the old applications's data files, so we clear the finder
files.
****************************************)
procedure initglobals
    (globals : wpointer);

var
    thepointer          :   longpointer;
    message             :   integer;
    count               :   integer;
    index               :   integer;

begin

    setglobal(long(globals));

    with globals^ do begin

        sysdir := systemdir(globals);

        thepointer
            := longpointer(iaznotify);
        iazaddr := thepointer^;
        launchaddr
            := BitAnd(long(@postlaunch),
                $FFFFFF);
    end;
    CountAppFiles(message, count);
    for index := 1 to count do
        ClrAppFiles(index);
end;
(****************************************)
end.

(****************************************)
/codetype PACK -15871 'Transfer HD' 16
packhead
pack
pas$library
macintfglue
/end

****************************************
*       rsrc.r
*
*       (c) 1987, 1988 Attic Software
*
*       resources for Transfer DA
*
****************************************

Transfer 3.2 HD
DFILDMOV

include drvr
```

```
include pack

*****************************************
*   Menu resource
*****************************************

type MENU
    ,-15871 (4)
Transfer
    About Transfer...
    Rebuild menu
    (-
    Finder

*****************************************
*
*   Error messages
*
*****************************************

Type STR#
    ,-15871 (32)
    4
I'm unable to load the Transfer DA.
Error building the menu list.
Error opening the "Transfer Data" file.
Error restoring the volume.

*****************************************
*
*   About information
*
*****************************************

Type INFO=GNRL
    Trans,-15870 (32)
.S
Transfer 3.2 HD adds a Transfer menu ++
to any application that ++
supports desk accessories.  It is ++
intended for use with your ++
hard disk, and
.S
expects that you are using ++
HFS.\0D\0DTransfer 3.2 HD was ++
written by Clifford Story of ++
Attic Software.\0D\0D
.S
                    Attic Software\0D++
                    P.O. Box 24695\0D++
                    Jacksonville, ++
Florida   32241

*****************************************
*
*   Item lists
*
*****************************************

Type DITL
    Error,-15871 (32)
2
*   1
BtnItem Enabled
116 83 136 173
OK

*   2
statText Disabled
10 10 106 246
^0^1\0D\0DTransfer HD error ^2,++
\0DSystem error ^3.

    About,-15870 (32)
2
*   1
BtnItem Enabled
252 201 272 291
OK

*   2
userItem Enabled
10 10 242 482

    Build,-15869 (32)
1
*   1
statText Disabled
10 10 90 246
Transfer is building the menu list.  ++
This will take 30 seconds ++
or so but the list will be preserved ++
and won't have to be ++
re-built unless you re-arrange your ++
disk.

    Delay,-15868 (32)
2
*   1
BtnItem Enabled
68 83 88 173
OK

*   2
statText Disabled
10 10 58 246
"^0" will be launched when you quit ++
this application.

*****************************************
*
*   Dialogs
*
*****************************************

Type DLOG
    Error,-15871 (32)
Transfer HD
64 128 210 384
invisible nogoaway
0
0
-15871

    About,-15870 (32)
Transfer 3.2 HD, © 1987, 1988 ++
Attic Software
50 10 332 502
invisible nogoaway
0
0
-15870

    Build,-15869 (32)
Transfer HD
64 128 164 384
invisible nogoaway
0
0
-15869

    Delay,-15868 (32)
Transfer HD
64 128 162 384
invisible nogoaway
0
0
-15868
```

© The Definitive MacTutor, Vol. 4

Advanced Mac'ing
IAC Driver Demo in MPW C



Frank Alviani
Contributing Editor
Waukegan, IL
MacTutor Vol. 4 No. 11

The Great IAC Sample Editor!! (or a reasonable facsimile thereof...) Frank Alviani

I had just crawled out of my sickbed and back to work. After getting the IAC Driver article to Dave Smith at MacTutor in the nick of time (as usual), wrestling with thousands of lines of Pascal was starting to look like a little rest - when who calls but our beloved assistant editor, Kirk.

"How's the next article coming, Frank? Gonna have it ready in time - maybe even a little early (wistful sigh)?" "No problem, Kirk", I assured him; I had already set aside the evening of the deadline to get it written. I would have preferred to be out dancing with Shellie and Alice like Paul, but an editor's gotta do what an editor's gotta do....

Before continuing with the actual meat of this article, please understand that I assume you have read (and hopefully understood) the previous IAC articles (in the August and October '88 issues of MacTutor). If you haven't, I'm afraid this is going to be quite incomprehensible...

The IAC Sample Editor was written to demonstrate how to implement most (though not all) of the capabilities provided by the SAWS IAC driver. I chose to use a text editor for the same reasons many others have chosen to write sample text editors - TextEdit handles the surprisingly groaty details of handling the characters, allowing me to concentrate on the other aspects of the application. It handles a few small "frills", such as choosing fonts and text sizes, because those are almost free; more complex enhancements such as scrolling have been left as "an exercise for the student". THE CRITICAL MATERIAL IS THE IAC HANDLING, NOT THE TEXT HANDLING. *[Window updating of the displayed text could use some work, but since we have published fully functional text editors in the past, the reader is referred to those articles for improvements to the text editing portion of the program. For purposes of this article, the ability to link documents betweeen two of these editors running together is what is important. A change in the linked text in one document, causes the linked text in another document to change automatically, thus creating integrated application software between two independent applications! -Ed]*

Like most software, this sample editor is not 100% perfectly debugged (my employer has this delusion that I should be producing code for him during the day rather than debugging software for MacTutor), but all the important pieces are reasonably functional. The functions that are known to have holes are:

1) Auto-linking between two previously linked documents is somewhat unreliable. Alternative methods of accomplishing this are discussed.

2) Identification of source and target extents in the "Links" menu is occasionally reversed.

Changes To The IAC Driver

During the course of debugging this sample editor, the "Complete Dependency" routine was found to be excessively complex and slightly buggy. After revision *[found on this month's source code disk, -Ed]*, the rules it follows for handling slot-IDs are much simpler:

1) Until a document registers as a data source, its slot is filled with a "dummy" value of 1 to hold its place and indicate "target only" status (it doesn't have a document ID yet).

2) When a target document does register as a data source, its slot is refilled with its assigned document ID.

In practice, this change would have very little effect on writing an application. The only real difference is that it is always a good idea to start a session with a slot-ID of 0 and let the IAC driver assign a slot, rather than trying to re-register under a slot-ID from the previous session.

Basic Additional IAC Necessities

The fundamental additional task facing an application using the IAC driver is that of keeping track of its data extents. You, as the source of data to other programs, must monitor each source of data in your document(s) and write any changed extents to the driver for the use of your client applications.

A change in a data extent can involve 2 factors: the contents of the extent, and its "size". Monitoring changes can vary widely in complexity, where most of the complexity involves monitoring the "size" of the data extents. For example, in an object-oriented graphics program such as MacDraw, the "size" of a data-extent (a graphic object) is unlikely to change under most circumstances, and so involves rather little effort to maintain normally. On the other hand, almost any operation in a text-processor is likely to change the size of an extent, and therefore involves quite a bit of effort to keep up-to-date.

Since most programs are at least as interested in using data as in generating it, you are likely to also be a client (or dependent) of one or more other programs. In this case, it is your responsibility to check with the driver periodically to see if any data sources you depend on have changed, and to down load the data if they have. You need to maintain the link "targets" so that they can be updated as new information becomes available.

Let's spend a few moments looking at how some basic tasks are done:

Checking for Updates - A client program is expected to periodically poll the IAC driver to determine if any of its source extents have been updated. Being a periodic action, this naturally fits into the main event loop. I recommend doing it during null-event processing, since you aren't interfering with other activity. A possible concern with this approach is that of loading down the system by continuously polling the driver, and slowing down the normal response time; in the sample editor, this polling is done only at 1 second intervals. The only negative to this is that there can be up to 1 second before client applications respond to an update.

Determining when to update the IAC driver - this is the server's side of the cooperation required. In the sample editor, I decided that the user would find the time needed to update the driver between keystrokes interfering with its usability, so the driver is updated only when you leave an extent [Note: in this version, only clicks outside an extent are checked - an extension for the future would be to also check the arrow keys.] In some applications (such as a real-time mail system) 1 update/keystroke might be exactly what is needed.

The ideal to strive for is to update the shared data as often as possible without sending meaningless intermediate data and without spending excessive time during the update process. Transmitting each digit to a spreadsheet as it is typed, for example, would be undesirable since recalculation might very well be triggered for each keystroke, rather than at the end when the entire number is sent.

Maintaining Extents - The simplest approach seems to do this on the fly. As mentioned above, changes to either the contents or the "size" of an extent need to be recorded. The sample editor in fact simply assumes that if you are leaving a source extent then you were probably doing something in it, and updates the driver. A more sophisticated approach would be to maintain a "dirty" flag for each extent, and update the IAC driver only if it was true.

Any operation that can affect the "size" of an extent determines the amount of change and adjusts the extent accordingly.

Providing a User Interface - In keeping with the visual nature of the Macintosh, there should be a visual method of making and displaying links between documents, that is consistent with the user interface guidelines Apple has developed. Simply typing in a starting and ending character position in a text editor, for example, is utterly unacceptable. The "magic clipboard" metaphor - a clipboard between applications that updates itself without intervention from the user - provides a logical extension that can be further developed.

User Interface Implementation

Most of the user-interface aspects of the IAC were implemented using menu commands. I chose this approach because it is familiar to all Mac users and independent of the application type.

All commands for starting and completing dependencies were placed on the Edit menu, since I am using the "magic clipboard" metaphor to make inter-application communications seem more familiar to the user. The "kill link" command is on the Edit menu simply because it is the location most of us would expect.

The method for displaying the location of IAC links is modeled after that of MPW: a menu with 1 menu item for each link in the document. The menu is not created until the first link exists. This approach was taken for several reasons. The most important implementation reason was that the ROM TextEdit package doesn't support discontinuous selections - if it did, a discontinuous selection together with an alternate highlighting routine would be terribly simple and could show all links at once. Lacking that, the menu seems familiar and straightforward; the only non-obvious detail is that the link will remain highlighted until the first mouse click.

A dialog box is used by the "Show Link Info" command to show the results of a census call, to find all the dependencies currently recorded by the IAC driver. Only the information returned by the census (document ID, hat check, and edition) is displayed; "size" and content info is likely to be incomprehensible to most applications other than the originator.

Fundamental Data Structures

In addition to the algorithms all programs implement, the other critical issue in programming is the selection of data structures. In an application supporting the IAC driver, 2 additional data structures need to be implemented beyond those needed for the application's work: a table of entries for each data extent (simply called the extent table from now on), and a small group of items that apply to a document as a whole, rather than to any particular individual extent.

Each entry in the extent table holds the "location" or "size" of the extent so that the application can determine when its contents have changed; how this is done is totally application-dependent. This "location" is also necessary so that you can determine when the user is entering and leaving an extent; as mentioned above this is important in deciding when to update the IAC driver. It is critical to record the source document_ID of an extent in order to know where you are the source or a target, and to identify the extent to the driver. The hat_check is also needed for identification. The last edition read of the extent is also needed so you can be sure to request its successor. A "dirty flag" would be a useful extension to further optimize sending data to the driver.

A certain amount of data needs to be kept about the document as a whole. This includes the slot_ID for the document, the document_ID, the number of extents for the document, and a handle to the extent table (although an application that would never handle more than one document could make the extent table a global data structure rather than allocating it on the heap).

Program Structure and Style

The IAC Sample Editor was developed in MPW C using MPW 2.0.2 on a 1-Megabyte MacPlus. This was done both because my development environment at home is identical to my environment at work, and because I am generally quite happy with MPW (with one caveat mentioned below).

The source code for the sample editor is broken into a number of files, for several reasons. The first reason is philosophical: working with a number of small, well-named files is much better when it comes to maintaining code. The second reason is pragmatic: this approach usually results in faster development, since far less of the source is processed when a change is made. The final reason is also pragmatic: it was the only way I could get the sample to build on my 1-Meg machine at home.

This brings up the caveat about MPW mentioned above - its memory appetite. The amount of memory needed for the C compiler seems rather excessive (especially compared to the Pascal compiler), and the overall requirements for MPW are high (1300K under MultiFinder, for example). While I am painfully aware of how much effort can be required to squeeze programs into smaller spaces, the cost of RAM is unlikely to drop a great deal in the near future, and requiring people to have 2-4 megabytes of RAM to use MPW 3.0 with source level debugging borders on arrogance. (End of tirade...)

Since this program was explicitly written as a sample to demonstrate how to use the IAC driver, it is somewhat more verbose than the traditional C program. I am convinced by experience that well-chosen names can be critical to the readability of code in any language, and try to write code that conveys as much meaning as possible at first glance. The argument that "C is a terse language" is simply concealed laziness; the slight extra effort involved in typing longer names will be repaid many times when you are debugging a routine, it is no longer fresh in your mind, and you don't need to rummage around in many different source files to try and recall what those obscurely name variables are really being used for.

Please note that the IAC routines are careful to return error codes for the use of the application. While this sample doesn't check them extremely thoroughly (nor does it check the ROM routine error codes as well as it should), a real applications should be paranoid about problems and pay attention to them!

The sample editor currently will support only 1 window. However, in order to simplify extension to multiple windows, all information relevant to an entire document (rather than just to individual extents) is maintained in a relocatable block whose handle is kept in the wRefCon field of the window record. It should not be very complex to extend the editor to properly support multiple windows.

The source files that make up the IAC Sample Editor, and their uses, are as follows:

Editor.b	The MPW script that controls the building process.
Editor.m	The input to "Make" that describes the dependencies between source files, and how to build any part of the Editor.
Editor.h	Globals and common definitions.
IAC.h	Definitions for the IAC interface package.
Editor.c	The mainline, and the extent-maintaining routines.
Editor.r	The Rez input file.
Boxes.c	Code for displaying the dialog boxes, including the census information.
Doc_rtns.c	Code for document management - open, close, save, etc.
Edit_IAC_rtns.c	Code for implementing "Hot Copy" and "Hot Paste" operations.
Edit_rtns.c	Code for implementing the standard text editing functions, with extensions for extent maintenance.
IAC.c	The interface package to the IAC driver.
SysEnvs.a	Glue for the SysEnvirons call.

While it would be tedious to read (and write!) a detailed description of every function in the program, certain functions do merit examination, since they demonstrate methods for using the IAC driver. Let's take a closer look at these routines.

Extent-maintaining Routines

The operations necessary to maintaining extent information have been collected into a group of routines which are in the main file Editor.c These are as follows:

chk_extent: This routine is used by all the other extent-maintaining functions. In this text editor, an extent is defined to be a character range (in others it could be a paragraph, for example). If the current selection overlaps an entry in the extent table, that extent is made the current extent. We check the current extent first, as a small optimization to try and avoid processing the entire extent table. In any case, we return a boolean that tells if we are now in an extent.

One limitation in this implementation is that it does not deal with a selection range that spans extents. In a real application an extent very likely would be a significant amount of data (rather than a single word, for instance), and a selection would be much less likely to span extents.

This is a function that is very application dependent, and will vary a lot in implementation, since the tests for a graphics program is utterly different from those for a text program.

ext_move: This is the routine that determines if the user has left the current extent, and updates the IAC driver if he/she has. If the selection is no longer in the original extent and that extent was a "source" extent (one for which we are sending data to the IAC driver), then we write the latest contents of that extent to the driver. We have nothing to do if we are transferring into an extent, or if the extent we are leaving is a "target" extent.

This does not maintain a "dirty" flag for each extent, and assumes that any source extent you are leaving has been changed, writing its data to the driver automatically. Maintaining a dirty flag for each extent (as well as for the document as a whole) would allow this routine to minimize traffic to the IAC driver.

ext_write: This routine is responsible for writing the contents of an extent to the IAC driver. It sets up a private buffer into which the data size, constant format of 'TEXT', and the data are written, and then just calls the interface routine iac_write_data.

ext_read: This routine is responsible for read the contents of an extent from the IAC driver. It automatically asks for the successor to the last extent read, and updates the extent with the edition actually returned by the driver. The selection range of the extent is selected and replaced by the new data from the driver.

If there was a change in the size of the extent, the extent is updated.

IAC User-Interface Routines

These are the routines that are the user's actual interface to the IAC driver. They are as follows:

do_hotCopy: This is the routine that establishes a "source" link. The first step is to check to see if we are already in an existing dependency; if we are, an alert is issued to the user pointing out that error, and we quit. Normal processing is to call iac_add_dependency to add an entry to the driver's dependency table. Since source documents are issued document IDs, this updates the doc_ID field of the window data block. Since the slot_ID field can be updated by the driver, it is also updated by this routine. A new entry is added to the table of extents associated with the document.

After the dependency has been successfully started, this routine also calls ext_write to send the initial copy of the contents of the extent to the driver (we assume that creating a link implies that another program will be immediately interested in the contents).

In order to give the user some way of viewing the links established between documents, add_display_cmd is called to add the newly established dependency source to the "Links" menu.

do_hotPaste: This is the routine that completes a link. Like do_hotCopy, the first step is to check to see if we are already in an existing dependency; if we are, an alert is issued to the user pointing out that error, and we quit. Normal processing is to call iac_complete_dependency. We do not update the document ID field since the document ID returned by the driver is that of the source document. Since the slot_ID field can be updated by the driver, it is updated by this routine. A new entry is added to the table of extents associated with the document.

After the dependency has been successfully completed, this routine calls ext_read to read the initial copy of the contents from the driver.

In order to give the user some way of viewing the links established between documents, add_display_cmd is called to add the newly established dependency target to the "Links" menu.

The two IAC routines use different algorithms for adding a new entry to the extent table, which was done for a reason. In some applications, the ordering of extents may not be important; do_hotCopy makes this assumption and just adds another extent to the end of the table. In other applications (such as a text editor) the order may be more relevant; do_hotPaste inserts a new entry so the starting locations of the extents are in ascending order. In this text editor, this means that only an extent and its successors need updating, rather than having to process the entire table. In a real application, of course, you need to take a consistent approach to maintaining the extent table.

Text Editing Routines

These perform the normal TextEdit functions, with the addition of the IAC code. Since development time was stretching out well beyond expectation, only the routine do_key actually updates extent-size information, but all of the routines that need to do so have the skeleton code in place, and all of them do actually communicate with the IAC driver.

do_clear: This uses the normal *TEDelete* call to the ROM to remove text without placing it on the clipboard, after calling chk_extent to see which , if any, extent is being affected. If an entire extent was cleared, iac_remove_dependency is called, otherwise iac_write_data is called to provide the latest data to the driver.

do_copy: This does nothing at all with the IAC driver, since it doesn't affect the data!

do_cut: The processing here is virtually identical to that in do_clear, except that *TECut* is called instead.

do_key: We check to see if we are in an extent (chk_extent), and then call *TEKey* to actually insert the character into the text. If we were in an extent, we only need to adjust the end of the extent. If a backspace was processed, we decrement the end, otherwise we increment it.

do_paste: This is very similar in processing to do_key. If we were in an extent, the end of the extent is incremented by the amount of text inserted. This is actually an oversimplification, since it doesn't take into account text replacement.

Document Maintaining Routines

These perform all the traditional file-system functions, with the addition of saving and restoring IAC information needed for the convenience of the user. They are not in alphabetical order:

setup_wind: This is a common routine is used to set up the window and initialize the extent table and window data structures.

save_doc: In addition to the normal function of writing the data to a file, creating it if necessary, this must also save the extent table and the document-wide data for future usage. This information is stored in resources of type 'EXTN'; if we are saving an existing file, previous resources are removed before writing the new versions.

open_doc: While opening a document is often a complex process due to the application's data structures, in the sample editor the complexity is due to the (optional) process of trying to reconnect any links that may have existed to already-open documents. This is half of the reconnection process; the other half, of having an open document reconnect to a just-opened document, would be a function of poll_iac.

The functions of this routine divide neatly into several parts:

(1) read the text data from a user selected file and create a window for it.

(2) retrieve the extent table and some of the document-wide data and attach them to the window.

(3) once these have been retrieved, try to re-establish any links that had existed previously, with already-opened documents. This is done as follows:

a) Issue an iac_census call to determine the current state of the universe. Assuming the call is successful and there are already documents that have posted their source dependencies,

b) Check each entry in the census against the extent table.

If a census entry matches an extent:

i) issue an iac_available_dependency call to setup the linkage,

ii) issue an iac_complete_dependency call to complete it,

iii) issue an ext_read call to get the latest copy of the data from the driver, and

iv) call add_display_cmd to add the dependency to the user's "Links" menu.

c) Now that as many as possible of your target extents are linked up again, it's time to post your source extents to the IAC driver. This is a simpler process, since it isn't necessary to match the census info against the extent table. For each extent just:

i) If you are the source for the extent, call iac_add_dependency to post the extent to the IAC driver,

ii) call ext_write to copy the latest version of your data up to the driver, and

iii) call add_display_cmd to add the dependency to the user's "Links" menu.

This is one approach to reconnecting documents that were linked in a "previous life". The other approach is roughly equivalent in complexity, but moves part (b) in the description above to poll_iac. Open_doc simply posts all of its source extents, and poll_iac recognizes any new sources that appear during a census. When a new source document appears, it then reconnects.

Other Routines

These serve important functions, but don't fit neatly into the other categories. They are as follows:

poll_iac: This is responsible for automatically downloading updated data from the IAC driver. It checks at 60-tick (1 second) intervals to balance timely updates against system traffic. iac_status is called to see if any changes at all have occurred.

There are several strategies that can be used to ensure that all updated links are read. The approach used here is simply to try and read every link for which it is a target, relying on the driver to ignore any extents which haven't been changed. Another, probably more efficient tack, is to make an iac_census call first. You would then check the edition of each source link against that stored in the extent's record, and only try and read those which you know have changed. The trade off is in the (slightly more complex) processing required for the census call against the (simpler) unnecessary read calls.

An extension mentioned above that can be added to this routine is to check the data returned by the iac_status call to see if a new document has added dependencies to the driver. If one has, you can then make an iac_census call to check the new entries against your extent table; should any of them be sources for your target links, those links can be re-established without further intervention. This is needed if an open document is to be able to re-link to a document opened after it.

link_display: This is used to highlight an extent. It uses a tiny assembler routine to implement an alternate highlighting routine as (obscurely) documented in the TextEdit chapter of Inside Macintosh. The routine is installed, and the extent range selected, which invokes our highlighting routine (which uses exclusive-or mode for reversibility). We then check the mouse button in order to wait for a mouse click to indicate the user is done, and reselect the original range (which calls our routine, which undoes the highlighting). Finally, the hook in the TE record is removed to restore TextEdit's normal behavior.

The only small difficulty came in getting the assembler routine linked in with the rest of the application. It turns out to be necessary to have the assembler's case option set to OBJECT so that the procedure label is written to the object file with the proper case (C is case sensitive), but to allow case-insensitivity for references to equate files. Sigh...

Debugging Concerns

Debugging under MultiFinder is slightly different than debugging under the normal Finder, primarily due to the ease with which you can get "lost" in memory. The latest version of TMON (2.8.1- which is what I always use for debugging), for example, has no facilities for dealing with multiple application zones. This is not the end of the world, however.

The primary tool at your disposal is attention: you must pay closer attention to the application zone than under the single-application system, since the only way to tell zones apart is by their address ranges. Fortunately, all of your regular debugging tools still work, and context switching occurs only at well-specified times, so life really isn't too bad once you get used to things.

The symbolic-debugging ability of TMON is virtually invaluable, and requires one small change to the normal C compilation. The "-g" option is required to have the MPW C compiler insert the stack frames and routine names needed by TMON, since the normal code generated doesn't require or use them.

It is often useful to set breakpoints in several applications at once. TMON supports this with no difficulty; the only trick is to be absolutely certain of which application is in the foreground when you set the breakpoint. I often start setting breakpoints in one program from the left end of the window, and breakpoints in the other program from the right end; this can sometimes make it a little easier to tell what's going on.

Final Notes

This has turned out to be a far larger piece of cake to chew than either Paul or I originally anticipated (merely proving again that all programmers are optimists, or we never would have gotten ourselves into all this...) and has also turned out to be very educational, to say the least. While this is the last article on the IAC I will be writing for a while (I am in the middle of massively renovating my house as this is being written), at least one more article that I know of should be appearing in the near future. Once my brain has cleared somewhat, there are several non-obvious extensions that can be implemented even with the driver in its current state, and a number of people have already discussed adding inter-machine communications. The future should be most interesting.

I cannot end this escapade without expressing my deepest gratitude to my infinitely-patient wife Alice, who has put up with a tired 1-year old by herself far too many times while I was up into the early morning trying to wring bugs out of stubborn code.

Listing: Editor.b

```
###    Editor.b - Build the sample editor
#
# Usage:
#    Set exit [0 | 1];
#    If exit == 0, all compiles and assemblies will be com-
pleted.  If any errors occur, subsequent steps will be
skipped. If exit == 1, the procedure stops after the first
error occurs.
#
#    Set hlxSrc "source directory"; Export hlsSrc
#    Set hlxEtc "result directory"; Export hlsEtc
#    Set hlxProg "auxiliary program"; Export hlsProg
#    general.b [ -r | -m] [makeParams…]
# # -r          Repeat previous build, using existing
"{hlxEtc}makeout". (Actually, this option merely suppresses
the "makeout" creation.)
# #             Must be the first (and only) parameter.
# # -m          Make "{hlxEtc}makeout", but do not execute it.
# #             Must be the first parameter.
# # makeParams Additional parameters to make; e.g.:
# #      -d pr="-p" Report progress
# #      -d e="#"   Suppress log detail
# #      -e         Make everything.
# #      -t         Merely touch dates to bring everything up-
to-date.

open "{log}"
#open "{worksheet}"
echo 'Date -a -t' 'Date -s -d' Start {hlxAlt} >> "{log}"

cd "{hlxSrc}"
set result Done
if "{1}" == "-m"
  set result Made
  shift 1
end
if "{1}" <> "-r"
  If {result} == Done
    echo set err 0 > "{hlxEtc}makeout"
    unset err
    make ("parameters") -f {hlxprg}.m >> "{hlxEtc}makeout"
  end || Set result Failed
end
if {result} == "Done"
  "{hlxEtc}makeout" || Set result Failed
end
echo 'Date -a -t' 'Date -s -d' {result} >> "{log}"
{MacAppDone}
exit 1 If result == Failed
```

Listing: Editor.m

```
# Make file for the Sample IAC-friendly Editor

ob = {hlxEtc}  #Set these definitions to your normal working
directories
sr = {hlxSrc}
prog = {hlx}
e  = echo

COptions = -o {ob} -g -q2
AOptions = -i {mpw}AIncludes -o "{ob}"

"{prog}Editor"    ƒ Editor.r {ob}Editor.code
  {e} 'Date -a -t' Rez Editor >> "{log}"
  Rez Editor.r -o "{prog}Editor" -t APPL -c IAC1
  Setfile -a b "{prog}Editor"    #set bundle bit

{ob}Editor.code ƒ {ob}Editor.c.o         {ob}IAC.c.o ∂
                  {ob}SysEnv.a.o         {ob}Doc_rtns.c.o ∂
                  {ob}Edit_IAC_rtns.c.o  {ob}Edit_rtns.c.o ∂
                  {ob}Editor.a.o         {ob}Boxes.c.o
  exit 1 if {err} <> 0
  {e} 'Date -a -t' Link Editor >> "{log}"
  Link {ob}Editor.c.o   {ob}IAC.c.o  {ob}SysEnv.a.o ∂
       {ob}Doc_rtns.c.o {ob}Edit_IAC_rtns.c.o
{ob}Edit_rtns.c.o ∂
       {ob}Editor.a.o   {ob}Boxes.c.o ∂
       "{CLibraries}"CRuntime.o ∂
       "{CLibraries}"CInterface.o ∂
       "{CLibraries}"StdCLib.o ∂
       "{CLibraries}"CSANELib.o ∂
       "{Libraries}"Interface.o ∂
       -t "OBJ " -o {ob}Editor.code

{ob}Editor.c.o ƒ {sr}Editor.c {sr}IAC.h  {sr}Editor.h
  {e} 'Date -a -t' C Editor >> "{log}"
  C {COptions} Editor.c || set err 1

{ob}IAC.c.oƒ {sr}IAC.c  {sr}IAC.h
  {e} 'Date -a -t' C IAC >> "{log}"
  C {COptions} IAC.c || set err 1

{ob}SysEnv.a.o ƒ {sr}SysEnv.a
  {e} 'Date -a -t' Asm SysEnv >> "{log}"
  Asm {AOptions} SysEnv.a || Set err 1

{ob}Editor.a.o ƒ {sr}Editor.a
  {e} 'Date -a -t' Asm Editor.a >> "{log}"
  Asm {AOptions} Editor.a || Set err 1

{ob}Doc_rtns.c.o     ƒ   {sr}Doc_rtns.c {sr}IAC.h
{sr}Editor.h
  {e} 'Date -a -t' C Doc_rtns >> "{log}"
  C {COptions} Doc_rtns.c || set err 1

{ob}Edit_IAC_rtns.c.o ƒ   {sr}Edit_IAC_rtns.c
{sr}IAC.h  {sr}Editor.h
  {e} 'Date -a -t' C Edit_IAC_rtns >> "{log}"
  C {COptions} Edit_IAC_rtns.c || set err 1

{ob}Edit_rtns.c.o     ƒ   {sr}Edit_rtns.c {sr}IAC.h
{sr}Editor.h
  {e} 'Date -a -t' C Edit_rtns >> "{log}"
  C {COptions} Edit_rtns.c || set err 1

{ob}Boxes.c.o         ƒ   {sr}Boxes.c{sr}IAC.h
{sr}Editor.h
  {e} 'Date -a -t' C Boxes >> "{log}"
  C {COptions} Boxes.c || set err 1
```

Listing: Editor.h

```
/*
 * Resource ID constants.
 */
# define appleID       128
# define fileID        129
# define editID        130
# define optionsID     131
# define fontID        132
# define sizeID        133
# define linkDisplayID 134

/* MyMenus[] array indexes */
# define appleMenu      0
# define aboutMeCommand 1

# define fileMenu      1
# define newCommand    1
```

```
# define openCommand  2
# define closeCommand 3
# define saveCommand  4
# define quitCommand  6

# define editMenu     2
# define undoCommand  1
# define cutCommand   3
# define copyCommand  4
# define pasteCommand 5
# define clearCommand 6

/* The IAC Commands!! */
# define hotCopyCommand  8
# define hotPasteCommand 9
# define zapLinkCommand  10

# define optionsMenu     3
# define showLinksCmd    1
# define showLinkInfoCmd 2

# define fontMenu        4
# define sizeMenu        5
# define linkdDispMenu   6
# define menuCount       7

/* windows, dialogs, and alerts */
# define windowID        128
# define ABOUT_DLOG      128
# define ABOUT_LINKS     129
# define NO_IAC          256
# define IAC_ERR_ALRT    257
# define NOT_IN_EXT      258
# define KILL_EXT        259
# define SAVE_CHANGES    260

# define BS 0x08
# define POLL_INT 60
# define SLEEP 20

/* Formats supported by IAC driver, normal resource types */
# define TXT_FMT 0x54455854

/* Error codes returned by the IAC driver */
# define NO_MORE_DOCS -2000
# define NO_MORE_SLOTS -2001
# define WRITE_FAILED -2002
# define MISSING_LINK -2003
# define NO_NEWER_ED -2004
# define READ_FAILED -2005
# define NO_SUCH_DEP -2006
# define OLD_ROMS -2007

/*
 *Information records associated with a window
 */

typedef struct {
  long   src_doc;       /* doc_ID of source doc */
  short    hat_check;   /* identifier for an extent */
  short    ed_level;    /* edition for extent */
  short    ext_strt;    /* start of extent range */
  short    ext_end;     /* end of extent range */
} extent, *extentP, *exTable, **extentH;

typedef struct {
  short    the_slot;    /* slot_ID for this document */
  long   doc_ID;        /* document ID for this doc */
  short    ext_cnt;     /* number of extents with this doc */
  short    relevent;    /* # un-updated extents */
  extentH    the_extents; /* block of extent records */
  TEHandle wind_TEH;      /* TEHandle for this window */
  Boolean    dirty;       /* doc needs saving */
  String(63) doc_file_nm; /* if null, never been saved */
} win_data, *win_dataP, **win_dataH;

/****
 * Global Data objects, used by routines external to main().
 ***/

#ifndef PUBLIC
#define PUBLIC
#undef NO_INIT
#else
#define NO_INIT
#endif

PUBLIC MenuHandle   MyMenus[menuCount]; /* The menu handles */
PUBLIC Boolean      DoneFlag;  /*true when File:Quit chosen */
PUBLIC TEHandle     TextH;          /* The TextEdit handle */
PUBLIC CursHandle   ibeamHdl;
PUBLIC WindowPtr    myWindow;     /* the text window */
PUBLIC WindowRecord   wRecord;    /* store the window stuff */
PUBLIC win_dataH    the_data_H;   /* data for a window */
PUBLIC short        fRef;       /* refnum for document file */
PUBLIC short      the_fNum, the_size;   /* text attributes */
PUBLIC Style      the_style;
PUBLIC extent     curr_ext;       /* latest extent worked on */
PUBLIC short      curr_ext_no;    /* index of curr_ext */
PUBLIC Boolean      ext_active;   /* an active extent? */
PUBLIC long       last_poll;      /* when we last polled IAC */

/*f_sizes & f_styles are used in the menu_tree() processing
and are read only */

#ifdef NO_INIT
PUBLIC short      extent_count;   /* links we are target of */
PUBLIC Boolean      link_menu;    /* link menu created */
PUBLIC short      f_sizes[8];
PUBLIC Style      f_styles[5];
#else
PUBLIC short      extent_count = 0;
PUBLIC Boolean      link_menu = false;
PUBLIC short      f_sizes[8] = {9, 10, 12, 14, 16, 20, 24,
28};
PUBLIC Style      f_styles[5] = {bold, italic, underline,
outline, shadow};
#endif

/*
 * HIWORD and LOWORD macros, for readability.
 */
# define HIWORD(aLong)  (((aLong) >> 16) & 0xFFFF)
# define LOWORD(aLong)  ((aLong) & 0xFFFF)
```

Listing: IAC.h

```
/*This is the set of externs needed to access the IAC inter-
face routines. The pointer/handle indicators are only remind-
ers */

extern short  iac_add_dependency();/* *doc_ID, *slot_ID,
*hat_check, *edition */
extern short  iac_available_dependency(); /* doc_id, hat_check
*/
extern short  iac_census(); /* *extent_count, **extent_info */
extern short  iac_complete_dependency(); /* *doc_id, *slot_id,
                                        *hat_check */
extern short  iac_open();
extern short  iac_read_data();  /* doc_id, slot_id, hat_check,
              *edition, fmt_pref[], *fmt_code, **ext_data */
extern short  iac_remove_dependency();   /* doc_id, slot_id,
                                      hat_check */
extern short  iac_status(); /* slot_id, *vers_id, *doc_count,
                          *extent_count */
extern short  iac_write_data(); /* doc_id, hat_check, *edition,
                          fmt_count, **ext_data */
```

```
/*error codes for iac_open, etc. */
# define NO_DRIVER -1
# define EARLY_SYS -2
# define MAX_EXTS 64

typedef struct {
    long   doc_ID;
    short    ed_level;
    short    hat_check;
} info_rec;

typedef struct {
    info_rec ext_entry[MAX_EXTS];
} info_tbl, *info_tblP, **info_tblH;
```

Listing: Editor.a

```
;*****************************************************************
;***** The SAWS Inter-Application Commun. Driver Sample Program
;***** Text Highlighting Routines for TextEdit
;***** Frank Alviani - 9/88
;*****************************************************************

    INCLUDE  'Traps.a'
    INCLUDE  'QuickEqu.a'
    INCLUDE  'SysEqu.a'
    INCLUDE  'SysErr.a'
    INCLUDE  'ToolEqu.a'
    STRING PASCAL
    CASE OBJECT

;
; This routine highlights the rectangle passed on the stack by
; "stippling" it. There could be separate routines for"target"
; and "source" links. Exclusive-or mode is used since this
; called twice (select and deselect) and we need to return the
; text to its original look.
; Input: A3 points to locked TERec
; Can destroy A0, A1, D0, D2, D3
;
SourceHigh PROC EXPORT
    MOVE.W #patXor,-(A7)   ;set mode
    _PenMode
    MOVE.L GrafGlobals(A5),A0
    PEA  ltGray(A0)    ;set stipple
    _PenPat
    _PaintRect      ;stipple the text
; _PaintRect pops rectangle off stack, leaving return address
    RTS

    END
```

Listing: Editor.c

```
/***
 *
 * File:   Editor.c
 *
 * Package:  Mainline
 *
 * Description: This is the mainline for the editor program to
test the Inter Application Communications (IAC) driver. Some
code from the Apple "sample.c" was adapted.
 *
 * Structure: In this source the following structure is used:
 *         includes & defines
 *         global variable definitions
 *         main()
 *         initialization routines
 *         the menu dispatcher menu_tree()
 *         dialog-box handlers
 *
 * Author:
 *FEA    6/88 - 7/88
 */

# include  <types.h>
# include  <memory.h>
# include  <quickdraw.h>
# include  <toolutils.h>
# include  <windows.h>
# include  <controls.h>
# include  <fonts.h>
# include  <events.h>
# include  <dialogs.h>
# include  <menus.h>
# include  <desk.h>
# include  <textedit.h>
# include  <segload.h>
# include  <string.h>
# include  <resources.h>

# include  <iac.h>
# undef PUBLIC
# include  <Editor.h>

#define noErr    0    /* 0 for success */

extern _DataInit();

/**
 * Routine: main()
 *
 *  Mainline of the test program
 */

int main()
{
short      init_result;    /* NULL if OK */
short      item;         /* alert button */
Rect       screenRect;
Rect       dragRect;
Rect       txRect;
Point      mousePt;
EventRecordmyEvent;
WindowPtr  theActiveWindow;
WindowPtr  whichWindow;

win_dataH  the_data_H;  /* data associated with a window */

extern void   menu_tree(), do_key(), poll_iac();
extern void   ext_move();
extern short  edit_init();
extern WindowPtr  myWindow;

  UnloadSeg(_DataInit);
  init_result = edit_init();
  if (init_result)
  {
    item = StopAlert(NO_IAC, nil);
    return 0;       /* allow C runtime cleanup */
  }

  UnloadSeg(edit_init);    /* if we get here, IAC open */

  /* window setup is handled by 'new' and 'open' commands */
  screenRect = qd.screenBits.bounds;
  SetRect(&dragRect, 4, 20 + 4, screenRect.right-4,
screenRect.bottom-4);

  /* The One True Event Loop */
  DoneFlag = false;
  for ( ;; )
  {
    if (DoneFlag)
    {
      break;   /* from main event loop */
    }
```

```
    /*
     * Main Event tasks:
     */

    theActiveWindow = FrontWindow();    /* Used often, avoid
repeated calls */

    if (myWindow && (myWindow == theActiveWindow))
    {
      GetMouse(&mousePt);
      SetCursor(PtInRect(&mousePt, &myWindow->portRect) ?
*ibeamHdl : &qd.arrow);
      TEIdle(TextH);

      the_data_H = (win_dataH) GetWRefCon (myWindow);
      if ((**the_data_H).dirty)
      {
        EnableItem (MyMenus[fileMenu], saveCommand);
      }
      else
      {
        DisableItem (MyMenus[fileMenu], saveCommand);
      }
    }

    if (!WaitNextEvent(everyEvent, &myEvent, SLEEP, nil))
    {
      /*A null or system event! IAC polling goes here.
        We only poll if we have link targets to check. */
      if (theActiveWindow && extent_count)
      {
        poll_iac(theActiveWindow);
      }
      continue;
    }

    the_data_H = (win_dataH) GetWRefCon(theActiveWindow);
    TextH = (**the_data_H).wind_TEH;

    switch (myEvent.what)
    {
      case mouseDown:
        switch (FindWindow(&myEvent.where, &whichWindow))
        {
          case inSysWindow:
            SystemClick(&myEvent, whichWindow);
            break;

          case inMenuBar:
            menu_tree(MenuSelect(&myEvent.where));
            break;

          case inDrag:
          DragWindow(whichWindow, &myEvent.where, &dragRect);
            break;

          case inGrow: /* There is no grow box. */
            break;

          case inContent:
            if (whichWindow != theActiveWindow)
            {
              SelectWindow(whichWindow);
            }
            else if (whichWindow == myWindow)
              {
                ext_move(&myEvent, theActiveWindow);
              }
            break;

          default:
            break;
        }/*endsw FindWindow*/
        break;

      case autoKey: /* ignore command-key */
        if (myWindow == theActiveWindow)
        {
          do_key(myEvent.message); /* check extents */
        }
        break;

      case keyDown:
        if (myWindow == theActiveWindow)
        {
          if (myEvent.modifiers & cmdKey)
          {
            menu_tree(MenuKey(myEvent.message & charCode-
Mask));
          }
          else
          {
            do_key(myEvent.message); /* check extents */
          }
        }
        break;

      case app4Evt: /* suspend/resume */
        break;

      case activateEvt:
        if ((WindowPtr) myEvent.message == myWindow)
        {
          if (myEvent.modifiers & activeFlag)
          {
            TEActivate(TextH);
            DisableItem(MyMenus[editMenu], undoCommand);
          }
          else
          {
            TEDeactivate(TextH);
            EnableItem(MyMenus[editMenu], undoCommand);
          }
        }
        break;

      case updateEvt:
        if ((WindowPtr) myEvent.message == theActiveWindow)
        {
          BeginUpdate(theActiveWindow);
          EraseRect(&theActiveWindow->portRect);
          TEUpdate(&theActiveWindow->portRect, TextH);
          EndUpdate(theActiveWindow);
        }
        break;

      default:
        break;

    } /* endsw myEvent.what */

  } /* for */

  if (myWindow)
  {
    /* shut down code, including "save changes?" */
    CloseWindow(myWindow);
  }
  return 0;     /* Return to allow C runtime cleanup */
}


 /**
 * Routine: chk_extent
 *
 *  This routine checks the selection range in the TE record
to determine if it intersects the current extent. If not, it
```

```
checks the entire set of extents to try and identify the extent
it might intersect. If we are notcurrently in an exent, the
'current extent' structure is filled with -1s as is
'curr_ext_no'. */

# define __SEG__ Main
Boolean  chk_extent(TextH, the_extH, ext_cnt)
  TEHandle   TextH;     /* The TextEdit record to check */
  extentH    the_extH;  /* handle to extent block */
  short      ext_cnt;   /* how many extents to process */
{
short      strt, endd;  /* range in TE record */
short      i;           /* scratch */
exTable    ext_recs;
Boolean    left, right;


  if (ext_cnt)
  {
    strt = (**TextH).selStart;
    endd = (**TextH).selEnd;
    ext_recs = *the_extH;

    /* Check last extent used. If not in that, scan entire
table. We are affecting an extent if EITHER the start or end
of the TE selection falls between the start and end of the
extent (inclusive).  */

    left = (curr_ext.ext_strt <= strt) && (curr_ext.ext_end >=
strt);
    right =  (curr_ext.ext_strt <= endd) && (curr_ext.ext_end
>= endd);
    if (left ||right) /* overlap on either end */
    {
      return (true);  /* existing info alright as is */
    }
    else   /* switched, check all */
    {
      for (i=0; i<ext_cnt; i++)
      {
        left = (ext_recs[i].ext_strt <= strt) &&
(ext_recs[i].ext_end >= strt);
        right = (ext_recs[i].ext_strt <= endd) &&
(ext_recs[i].ext_end >= endd);
        if (left ||right) /* this now the current extent */
        {
          curr_ext.ext_strt =  ext_recs[i].ext_strt;
          curr_ext.ext_end  =  ext_recs[i].ext_end;
          curr_ext.hat_check = ext_recs[i].hat_check;
          curr_ext.ed_level =  ext_recs[i].ed_level;
          curr_ext_no = i;
          return (true);
        }
      } /* end for */
    } /* end else */
  } /* end there-are-extents */

  curr_ext.ext_strt =  -1; /* not current in an extent */
  curr_ext.ext_end  =  -1;
  curr_ext.hat_check = -1;
  curr_ext.ed_level =  -1;
  curr_ext_no = -1;
  return(false);
}

/**
 * Routine: ext_write
 *
 *  This routine handles the mechanics of actually writing an
extent's data to the IAC driver. */

# define __SEG__ Main
short  ext_write(TextH, strt, sz, the_doc, the_hatcheck,
the_ed)
  TEHandle TextH;     /* The TextEdit handle */
  short    strt;      /* position of 1st char to write */
  short    sz;        /* count of bytes */
  long     the_doc;   /* which doc */
  short    the_hatcheck;
  short     *the_ed;
(
short    iac_code;       /* result from IAC call */
long     data_size;
Handle   ext_data, t_base;
long     *l_ptr;         /* recast to ease setting up hdr */

# define   HDR_SIZE 8

  ext_data = NewHandle (sz + HDR_SIZE);
  l_ptr = (long *)(*ext_data);
  *l_ptr = TXT_FMT;
  *(l_ptr+1) = (long) sz;
  t_base = (**TextH).hText;
  BlockMove (*t_base + strt,        /* source */
         (*ext_data) + HDR_SIZE,    /* dest */
         (long) sz);
  iac_code = iac_write_data(the_doc, the_hatcheck, the_ed, 1,
ext_data);
  DisposHandle(ext_data);
  return (iac_code);
}

/**
 * Routine: ext_read
 *
 *  This routine handles the mechanics of actually reading an
extent's data from the IAC driver. If successful the
edition_level for the extent is updated. */

# define __SEG__ Main
short  ext_read(w_Ptr, the_ed, which)
  WindowPtr  w_Ptr;
  short      *the_ed;    /* edition of the extent to read */
  short      which;      /* which extent (zero-based) */
{
win_dataH  the_data_H;   /* data associated with a window */
extentH    the_extH;     /* handle to extent block */
TEHandle   TextH;        /* The TextEdit handle */
exTable    ext_recs;     /* walking extents for adjusting */
long     fmt_code;       /* actual data format from IAC */
long     fmt_pref[3];    /* format prefs, descending  */
long     old_st, old_end;  /* TE selection before poll */
short    delta;        /* change in extent size */
short    an_ed, j;     /* traditional loop counters */
short    data_size;
Handle   ext_data;     /* Handle to data block */

short    iac_code = noErr; /* result from IAC call */

  the_data_H = (win_dataH) GetWRefCon(w_Ptr);
  TextH =      (**the_data_H).wind_TEH;
  the_extH =  (**the_data_H).the_extents;

  fmt_pref[0] = TXT_FMT;
  fmt_pref[1] = 0;
  HLock(the_extH);
  ext_recs = *the_extH;
  ext_data = NewHandle(0L);
  /* need memory test here */

  an_ed = *the_ed + 1;
  iac_code = iac_read_data(ext_recs[which].src_doc,
              (**the_data_H).the_slot,
              ext_recs[which].hat_check,
              &an_ed,
              &fmt_pref[0],
              &fmt_code,
              ext_data);
```

```
  if (iac_code == noErr)
  {
    ext_recs[which].ed_level = an_ed; /* update extent */
    *the_ed = an_ed;          /* update caller */
    data_size = GetHandleSize(ext_data);

    delta = data_size - (ext_recs[which].ext_end -
              ext_recs[which].ext_strt);

    TESetSelect (ext_recs[which].ext_strt,  /* select extent
for TE */ ext_recs[which].ext_end,TextH);
    TEDelete(TextH);
    HLock(ext_data);
    TEInsert(*ext_data, data_size, TextH);/* update text */
    HUnlock(ext_data);

    ext_recs[which].ext_end = ext_recs[which].ext_strt +
data_size; /* update end (start unchanged) */

    if (delta) /* only update if there was a change */
    {
      /* offset each following extent by change in size of
this extent */
      for (j=which+1; j<(**the_data_H).ext_cnt; j++)
      {
        ext_recs[j].ext_strt += delta;
        ext_recs[j].ext_end +=  delta;
      }
    }
  }

  HUnlock(the_extH);
  DisposHandle (ext_data);   /* clean up */
  return (iac_code);
}

/**
 * Routine: ext_move
 *
 *  This routine checks to see if we have left the "current
extent". Ifwe have, we check to see if we are the source for
that extent. If so, the contents of that extent are written to
the IAC driver. If we are instead mousing INTO an extent, that
becomes the current extent.
 */

# define __SEG__ Main
void ext_move(myEvent, w_Ptr)
  EventRecord*myEvent;
  WindowPtr    w_Ptr;
{
EventRecord  anEvent;
short      iac_code;    /* result from IAC call */
OSErr      an_err;
win_dataH  the_data_H;  /* data associated with a window */
extentH    the_extH;    /* handle to extent block */
exTable    ext_recs;    /* for updating old extent */
TEHandle   TextH;       /* The TextEdit handle */
short      i, the_ed, old_ext_no;
short    data_size;
extent   old_ext;     /* to check for change in extents */
Handle   ext_data, t_base;
long     *l_ptr;      /* recast to ease setting up hdr */

extern Boolean ext_active;
extern Boolean  chk_extent();

#define    HDR_SIZE 8

  the_data_H = (win_dataH) GetWRefCon(w_Ptr);
  TextH =     (**the_data_H).wind_TEH;
  the_extH =  (**the_data_H).the_extents;
  ext_recs =  *the_extH;
  old_ext_no = curr_ext_no;
  anEvent =   *myEvent;   /* local copy for modification */

  GlobalToLocal(&anEvent.where);
  TEClick(&anEvent.where, (anEvent.modifiers & shiftKey) != 0,
TextH);

  if (ext_active) /* we may need to write it's data to the IAC
driver */
  {
    old_ext = curr_ext;
    the_ed = curr_ext.ed_level;
    ext_active = chk_extent(TextH, the_extH,
(**the_data_H).ext_cnt);

    if ((old_ext.ext_strt != curr_ext.ext_strt) ||
      !ext_active) /* we left the current extent */
    {
      if (curr_ext.src_doc==(**the_data_H).doc_ID)
      /* we were the source */
      {
        iac_code = ext_write(TextH,
                  old_ext.ext_strt,
                  (old_ext.ext_end - old_ext.ext_strt),
                  (**the_data_H).doc_ID,
                  old_ext.hat_check,
                  &the_ed);
        ext_recs[old_ext_no].ed_level = the_ed;
        /* update old extent just written */
      }
    }
  }
  else /* see if we've moved into one! */
  {
    ext_active = chk_extent(TextH, the_extH,
(**the_data_H).ext_cnt);
    /* Since we weren't in an extent, nothing to write */
  }
}

/**
 * Routine: ext_remove
 *
 *  This routine severs a link in the drivers dependency
table. It assumes the driver will take care of removing any
remaining data it sent. */

# define __SEG__ Main
void ext_remove(w_Ptr, extP)
  WindowPtr    w_Ptr;
  extentP      extP;      /* extent to remove */
{
win_dataH  the_data_H;  /* data associated with a window */
extentH    the_extH;    /* handle to extent block */
short      iac_code;    /* result from IAC call */

  the_data_H = (win_dataH) GetWRefCon (w_Ptr);
  iac_code = iac_remove_dependency((**the_data_H).doc_ID,
            (**the_data_H).the_slot, extP->hat_check);
}

/**
 * Routine: poll_iac
 *
 *  This routine checks with the IAC driver to see if there's
anything to do. It checks at 1 second intervals to avoid
excessive loading onsystem capacity. In order to simplify
management of the extent ranges, extents are stored in ascend-
ing order by starting position. This means that updating an
extent involves adjusting the start/end of all SUCCEEDING
extents by the amount the updated extent changes, but no
preceeding extents need be touched. Storing the extents in
random order is definitely a poor idea. */
```

```
# define __SEG__ Main
void poll_iac(w_Ptr)
  WindowPtr     w_Ptr;
{
win_dataH  the_data_H;  /* data associated with a window */
extentH    the_extH;    /* handle to extent block */
exTable    ext_recs;    /* for walking extents to read */
TEHandle   TextH;       /* The TextEdit handle */
short      slot_ID, vers, the_ed;/* for IAC */
long       fmt_code;    /* actual data format from IAC */
long fmt_pref[3]; /* format prefs, descending desirability */
long  old_st, old_end;  /* TE selection before poll */
long  my_docID;       /* so I don't read my own extents! */
short delta;          /* change in extent size */
short i,j;            /* traditional loop counters */
Handle   ext_data;    /* Handle to data block */

short  iac_code = noErr; /* result from IAC call */
short  waiting = 0;      /* for IAC */
short  doc_count = 0;    /* how many are open? */

extern long last_poll;    /* when we last polled IAC */

  if (TickCount() > (last_poll+POLL_INT)) /* min interval */
  {
    /* get relevent handles, etc */
    the_data_H = (win_dataH) GetWRefCon (w_Ptr);
    slot_ID =   (**the_data_H).the_slot;
    the_extH =  (**the_data_H).the_extents;
    TextH =     (**the_data_H).wind_TEH;
    my_docID =  (**the_data_H).doc_ID;

    iac_code = iac_status(slot_ID, &vers, &doc_count, &wait-
ing);

    if (waiting>0) /* get data for each and update extents */
    {
      old_st =   (**TextH).selStart; /* user's selection */
      old_end =  (**TextH).selEnd;
      ext_recs = *the_extH;       /* point to 1st extent */
      for (i=0; i<(**the_data_H).ext_cnt; i++)
      {
        if (my_docID != ext_recs[i].src_doc) /*not my own! */
        {
          the_ed = ext_recs[i].ed_level + 1; /* successor to
last level read */
          iac_code = ext_read(w_Ptr, &the_ed, i);

          if (iac_code==noErr)
          {
            if ((waiting -= 1) == 0)
            {
              break; /* stop if no extents left to do */
            }
          }
        }
      } /* for */

      TESetSelect (old_st, old_end, TextH); /* restore */
    }

    last_poll = TickCount(); /* update timer */
  }
}

/**
 * Routine: edit_init
 *
 *  This handles all the grunt initialization. It returns NULL
if everything went OK, non-NULL if unable to open the IAC
driver or we aren't operating under MultiFinder. */

# define __SEG__ init
short edit_init()
{
short    i, k, menu_limit;
Str255   itemString;

short    result = 0;  /* default optimism */

extern MenuHandle MyMenus[];
extern CursHandle ibeamHdl;
extern long   last_poll;

  InitGraf(&qd.thePort);
  InitFonts();
  FlushEvents(everyEvent, 0);
  InitWindows();
  InitMenus();
  TEInit();
  InitDialogs(nil);
  InitCursor();
  ibeamHdl = GetCursor(iBeamCursor);
  HNoPurge((Handle)ibeamHdl);    /* ensure we keep it */

  result = iac_open();/* try to get IAC driver */
  if (result != noErr)/* couldn't get it! */
  {
    NumToString ((long) result, &itemString);
    if (result==EARLY_SYS)
    {
      ParamText ("MultiFinder","",&itemString,"");
    }
    else
    {
      ParamText ("the IAC driver","",&itemString,"");
    }
  }
  else         /* got it! */
  {
    ext_active = false;     /* no "current extent" yet */
    curr_ext.hat_check = 0;
    curr_ext.ed_level =  0;
    curr_ext.ext_strt =  0;
    curr_ext.ext_end =   0;

    /* handle menu init'g (don't add link-display menu) */
    menu_limit = menuCount-1;
    for (i=appleMenu,k=appleID; i<menu_limit; i++,k++)
    {
      MyMenus[i] = GetMenu(k);
    }
    AddResMenu(MyMenus[appleMenu], (ResType) 'DRVR');
    AddResMenu(MyMenus[fontMenu], (ResType) 'FONT');
    for (i=0; i<menu_limit ;++i )
    {
      InsertMenu(MyMenus[i],0);
    }

    /* set textstyling defaults */
    GetFNum ("Geneva", &the_fNum);
    the_size = 10;
    CheckItem (MyMenus[sizeMenu], 2, true);
    /* scan font menu to check default font */
    for (i=1; i<=CountMItems (MyMenus[fontMenu]); i++)
    {
      GetItem (MyMenus[fontMenu], i, &itemString);
      GetFNum (&itemString, &k);
      if (k==geneva)
        {
          CheckItem (MyMenus[fontMenu], i, true);
          break;
        }
    }
    DrawMenuBar();        /* now user can see menu bar */
    last_poll = TickCount(); /* so we can poll @ 1-sec
intervals */
  }
```

```
  return(result);
}

/**
 * Routine: menu_tree
 *
 *   This is the standard menu-processing tree.
 */

# define __SEG__ Main
void menu_tree(menu_sel)
  long menu_sel;  /* menu/item selected */
{
short    the_menu, the_item;
GrafPtr  savePort;
char     daName[256];
Str255   itemString;     /* font selection */
short    i;
win_dataH  the_data_H;  /* data associated with a window */
extentP    extP;  /* ptr to an extent to zap on quitting */
TEHandle   TextH;       /* The TextEdit handle */

extern short    the_fNum, the_size;   /* current attributes */
extern Style    the_style;
extern MenuHandle MyMenus[];
extern Boolean    DoneFlag;

extern void  about_box();
extern short open_doc(), create_doc(), close_doc(),
  save_doc();
extern void  do_clear(), do_copy(), do_cut(), do_paste();
extern void  do_hotCopy(), do_hotPaste(), killLink_box(),
  ext_remove();
extern void  link_display(), linkInfo_box();

  the_item = LOWORD(menu_sel);
  the_menu = HIWORD(menu_sel); /* This is the resource ID */

  if (myWindow)
  {
    the_data_H = (win_dataH) GetWRefCon (myWindow);
    TextH = (**the_data_H).wind_TEH;
  }

  switch (the_menu)
  {
    case appleID:
      if (the_item == aboutMeCommand)
      {
        about_box();
      }
      else
      {
        GetItem(MyMenus[appleMenu], the_item, daName);
        GetPort(&savePort);
        (void) OpenDeskAcc(daName);
        SetPort(savePort);
      }
      break;

    case fileID:
      switch (the_item)
      {
        case newCommand:
          (void) create_doc();
          break;

        case openCommand:
          (void) open_doc();
          break;

        case closeCommand:
          (void) close_doc();
          break;

        case saveCommand:
          (void) save_doc();
          break;

        case quitCommand:
          DoneFlag = true;      /* I want out! */

          for (i=0; i<(**the_data_H).ext_cnt; i++)
          {
            extP = (*(**the_data_H).the_extents) +
                 (i * sizeof(extent));
            ext_remove(myWindow, extP);
          }
          break;

        default:
          break;
      }
      break; /* fileID */

    case editID:
      switch (the_item)
      {
        case undoCommand: /* not implemented (if ever!) */
          break;

        case cutCommand:
          do_cut();
          break;

        case copyCommand:
          do_copy();
          break;

        case pasteCommand:
          do_paste();
          break;

        case clearCommand:
          do_clear();
          break;

        case hotCopyCommand:
          do_hotCopy();
          break;

        case hotPasteCommand:
          do_hotPaste();
          break;

        case zapLinkCommand:
          killLink_box();
          break;

        default:
          break;
      }
      break; /* editID */

    case optionsID:
      switch (the_item)
      {
        case showLinksCmd:
          break;

        case showLinkInfoCmd:
          linkInfo_box();
          break;

        default:
          break;
      }
```

```
      break; /* optionsID */

    case fontID:
      for (i=1; i<=CountMItems(MyMenus[fontMenu]); i++)
      {
        CheckItem (MyMenus[fontMenu], i, false);
      }
      CheckItem (MyMenus[fontMenu], the_item, true);
      GetItem (MyMenus[fontMenu], the_item, &itemString);
      GetFNum (&itemString, &the_fNum);
      for (i=0; i<8; i++)   /* set real sizes */
      {
        if (RealFont (the_fNum,f_sizes[i]))
        {
          SetItemStyle (MyMenus[sizeMenu], i+1, outline);
        }
        else
        {
          SetItemStyle (MyMenus[sizeMenu], i+1, normal);
        }
      }
      if (myWindow)/* force a refresh */
      {
        SetPort (myWindow);
        TextFont (the_fNum);
        (**TextH).txFont = the_fNum;
        TECalText (TextH);
        InvalRect (&myWindow->portRect);
      }
      break;

    case sizeID:
      if (the_item<9)   /* setting new size */
      {
        for (i=1; i<9; i++)   /* remove all checkmarks */
        {
          CheckItem (MyMenus[sizeMenu], i, false);
        }
        the_size = f_sizes[the_item-1];
        CheckItem (MyMenus[sizeMenu], the_item, true);
      }
      else       /* setting style */
      {
        if (the_style & f_styles[the_item-10])/*already on?*/
        {
          the_style &= (~f_styles[the_item-10]);/*turn off*/
          CheckItem (MyMenus[sizeMenu], the_item, false);
        }
        else
        {
          the_style |= f_styles[the_item-10];
          CheckItem (MyMenus[sizeMenu], the_item, true);
        }
      }
      if (myWindow)/* force a refresh */
      {
        SetPort (myWindow);
        TextFace (the_style);
        TextSize (the_size);
        (**TextH).txFace = the_style;
        (**TextH).txSize = the_size;
        TECalText (TextH);
        InvalRect (&myWindow->portRect);
      }
      break;

    case linkDisplayID:
      link_display(the_item - 1);/* zero-based extent sub-
scripts */
      break;

    default:
      break;
  } /* end switch(the_menu) */

  HiliteMenu(0);

  return;
}

/**
 * Routine: add_display_cmd()
 *
 *  This is the routine that is called whenever a link is
completed. It adds a menu item to the "Links" menu so the user
can highlightany visible links in a distinctive fashion. */

# define __SEG__ Main
void add_display_cmd(which, total)
  short  which;   /* position in extent array */
  short  total;   /* total # of extents */
{
win_dataH  the_data_H;  /* data associated with a window */
extentH    the_extH;    /* handle to extent block */
exTable    ext_recs;
Str255     num_str, cmd_str; /* for building menu item */

  the_data_H = (win_dataH) GetWRefCon (myWindow);
  the_extH =  (**the_data_H).the_extents;
  ext_recs =  *the_extH;

  if ((**the_data_H).ext_cnt)
  {
    if (!link_menu)   /* need to create link menu */
    {
      MyMenus[linkdDispMenu] = GetMenu(linkDisplayID);
      InsertMenu(MyMenus[linkdDispMenu],0);
      link_menu = true;
    }

    NumToString ((long) total, &num_str);
    if (ext_recs[which].src_doc == (**the_data_H).doc_ID)
    {
      strcpy(&cmd_str, "Source extent ");
    }
    else
    {
      strcpy(&cmd_str, "Target extent ");
    }
    (void) strcat(&cmd_str, &num_str);  /* build menu item */
    AppendMenu (MyMenus[linkdDispMenu], &cmd_str);

    DrawMenuBar();    /* now user can see menu bar again */
  }
}

# define __SEG__ Main
void link_display(ext_no)
  short    ext_no;    /* which extent to highlight */
{
win_dataH  the_data_H;  /* data associated with a window */
extentH    the_extH;    /* handle to extent block */
exTable    ext_recs;        /* ptr to array of extents */
short      old_st, old_end;  /* previous selection range */
TEHandle   TextH;           /* The TextEdit handle */

extern voidSourceHigh(); /* assembler highlighting routine */

  the_data_H = (win_dataH) GetWRefCon (myWindow);
  the_extH =  (**the_data_H).the_extents;
  TextH =     (**the_data_H).wind_TEH;
  ext_recs =  *the_extH;

  old_st =  (**TextH).selStart;
  old_end = (**TextH).selEnd;

  /* put a routine address into teHiHook for new highlighting
routine */
```


© The Definitive MacTutor, Vol. 4

```
  (**TextH).highHook = &SourceHigh();
  TESetSelect ((long) ext_recs[ext_no].ext_strt,
        (long) ext_recs[ext_no].ext_end,TextH);

  while (!Button()) ;    /* wait for mouse-down */
  while (Button()) ;     /* and following mouse-up */

  /* clear teHiHook to restore normal highlighting */
  TESetSelect ((long) old_st, (long) old_end, TextH);
  /* unhighligh extent */
  (**TextH).highHook = nil;
}
```

Listing: Editor.r

```
/*resource file for the test editor */

#include "Types.r"

/* - The Menus - */

resource 'MENU' (128, "Apple", preload) {
  128,
  textMenuProc,
  0x7FFFFFFD,/* Disable item #2 */
  enabled,
  apple,
  {
    "About Sample…",
      noicon, nokey, nomark, plain;
    "-",
      noicon, nokey, nomark, plain
  }
};

resource 'MENU' (129, "File", preload) {
  129,
  textMenuProc,
  0x2F,
  enabled,
  "File",
  {
    "New",
      noicon, "N", nomark, plain;
    "Open",
      noicon, "O", nomark, plain;
    "Close",
      noicon, noKey, nomark, plain;
    "Save",
      noicon, "S", nomark, plain;
    "-",
      noicon, nokey, nomark, plain;
    "Quit",
      noicon, "Q", nomark, plain
  }
};

resource 'MENU' (130, "Edit", preload) {
  130, textMenuProc,
  0x7FFFFFBD,/* Disable items #1 & #2 */
  enabled, "Edit",
   {
    "Undo",
      noicon, "Z", nomark, plain;
    "-",
      noicon, nokey, nomark, plain;
    "Cut",
      noicon, "X", nomark, plain;
    "Copy",
      noicon, "C", nomark, plain;
    "Paste",
      noicon, "V", nomark, plain;
    "Clear",
      noicon, nokey, nomark, plain;
    "-",
      noicon, nokey, nomark, plain;
    "Hot Copy",
      noicon, nokey, nomark, plain;
    "Hot Paste",
      noicon, nokey, nomark, plain;
    "Remove Link",
      noicon, nokey, nomark, plain
  }
};

resource 'MENU' (131,"OptionsMenu",Preload) {
131,     /* ID */
textMenuProc,/* menu def proc ID */
0B11,          /* item flags */
enabled,      /* menu enable */
"Options",
  { "Show links",
    noIcon,noKey,noMark,plain;
    "Show link info",
    noIcon,noKey,noMark,plain
  }
};

resource 'MENU' (132,"FontMenu",Preload) {
132,     /* ID */
textMenuProc,/* menu def proc ID */
allEnabled,  /* item flags */
enabled,      /* menu enable */
"Font",
(
)
};

resource 'MENU' (133,"Sizes",Preload) {
133,     /* ID */
textMenuProc,/* menu def proc ID */
0B11111011111111,   /* item flags */
enabled,      /* menu enable */
"Size",
  ( "9",
    noIcon,noKey,noMark,plain;
    "10",
    noIcon,noKey,noMark,plain;
    "12",
    noIcon,noKey,noMark,plain;
    "14",
    noIcon,noKey,noMark,plain;
    "16",
    noIcon,noKey,noMark,plain;
    "20",
    noIcon,noKey,noMark,plain;
    "24",
    noIcon,noKey,noMark,plain;
    "28",
    noIcon,noKey,noMark,plain;
    "-",
    noIcon,noKey,noMark,plain;
    "Bold",
    noIcon,"B",noMark,plain;
    "Italic",
    noIcon,"I",noMark,plain;
    "Underlined",
    noIcon,"U",noMark,plain;
    "Outline",
    noIcon,noKey,noMark,plain;
    "Shadow",
    noIcon,noKey,noMark,plain
  )
};

resource 'MENU' (134,"Links",Preload) {
134,     /* ID */
textMenuProc,/* menu def proc ID */
```

```
0xFFFFFFFF,  /* item flags */
enabled,     /* menu enable */
"Links",
{
}
};

/* - The Windows - */

resource 'WIND' (128,"a Window") {
  {64, 60, 314, 460},
  documentProc,
  visible,
  noGoAway,
  0x0,
  "Untitled"
};

/* - The Alerts/Dialogs (with DITLs) - */

/* 128 is the about box */
/* 129 is the link info box */

/* 256 is "No IAC" error box */
/* 257 is the IAC error alert */
/* 258 is the "Not in any extents" alert */
/* 259 is the "Sure you want to kill this extent?" Alert */
/* 260 is the "Save changes?" alert */

resource 'DLOG' (128,Purgeable) {
  {61, 80, 301, 420},
  altDBoxProc,
  visible,
  noGoAway,
  0x0,
  128,
  "x"
};

resource 'DITL' (128,Purgeable) {
  {
    {200, 110, 225, 230},    /* [1] */
    Button {enabled, "Wonderful!"},
    {15, 65, 31, 275},       /* [2] */
    StaticText {disabled,"The Great IAC Sample Program!"},
    {40, 20, 56, 320},       /* [3] */
    StaticText {disabled,"This copy: ^0"},
    {60, 20, 124, 320},      /* [4] */
    StaticText {disabled,"This sample is designed to show off
the "
      "var- ious capabilities of the  SAWS Inter- Application
"
      "Communications Driver. It may be used freely. Send
questions, etc"
      ", to: "},
    {124, 119, 140, 219},    /* [5] */
    StaticText {disabled,"Frank Alviani"},
    {140, 120, 188, 220},    /* [6] */
    StaticText {disabled,"P.O. Box 8744 Waukegan Ill 60079"}
  }
};


resource 'DLOG' (129,"link info",Purgeable) {
  {80, 86, 300, 426},
  altDBoxProc,
  visible,
  noGoAway,
  0x0,
  129,
  ""
};

resource 'DITL' (129,Purgeable) {
  { {185, 130, 210, 210},    /* [1] */
    Button {enabled,"OK!"},
    {10, 20, 26, 320},       /* [2] */
    StaticText {disabled,"Information About Current Registered
Links"},
    {35, 300, 170, 316},   /* [3] */
    UserItem {enabled},
    {35, 20, 170, 300},      /* [4] */
    UserItem {disabled}
  }
};

/* Scroll bar for link-info dlog */
resource 'CNTL' (129,Purgeable) {
  {35,300,170,316},
  0, /* value */
  visible,
  0, /* max */
  0, /* min */
  scrollBarProc,  /* type */
  0, /* refcon */
  "x"
};


resource 'ALRT' (256,Purgeable) {
  {100, 100, 220, 400},
  256,
  { OK, visible, sound1,/* [1] */
    OK, visible, sound1,/* [2] */
    OK, visible, sound1,/* [3] */
    OK, visible, sound1   /* [4] */
  }
};

resource 'DITL' (256,Purgeable) {
  { {90, 200, 110, 280},/* [1] */
    Button {enabled, "Rats!" },
    {20, 72, 84, 280},     /* [2] */
    StaticText {disabled,
      "Unfortunately, we cannot find ^0. "
      "Without it, there is no point in going on. Fare-
well..."};
    {94,20,110,150},       /* [3] */
    StaticText {disabled, "ID=^2"}
  }
};

resource 'ALRT' (257,Purgeable) {
  {100, 100, 240, 400},
  257,
  { OK, visible, sound1,/* [1] */
    OK, visible, sound1,/* [2] */
    OK, visible, sound1,/* [3] */
    OK, visible, sound1   /* [4] */
  }
};

resource 'DITL' (257,Purgeable) {
  { {105, 110, 130, 190},    /* [1] */
    Button {enabled,"Rats!"},
    {10, 67, 90, 280},       /* [2] */
    StaticText {disabled,"IAC Error ^0 was encountered during
the "
      "^1 operation, which was not completed. Sorry!")
  }
};

resource 'ALRT' (258,Purgeable) {
  {100, 100, 200, 400},
  258,
  { OK, visible, sound1,/* [1] */
    OK, visible, sound1,/* [2] */
    OK, visible, sound1,/* [3] */
```

```
    OK, visible, sound1   /* [4] */
  }
};

resource 'DITL' (258,Purgeable) {
  { {70, 110, 95, 190},   /* [1] */
    Button {enabled,"Rats!"},
    {10, 62, 58, 290},    /* [2] */
    StaticText {disabled,"Sorry! You are not in any extent."
      "Therefore no link can be severed."}
  }
};

resource 'ALRT' (259,Purgeable) {
  {100, 100, 200, 400},
  259,
  { OK, visible, sound1,/* [1] */
    OK, visible, sound1,/* [2] */
    OK, visible, sound1,/* [3] */
    OK, visible, sound1   /* [4] */
  }
};

resource 'DITL' (259,Purgeable) {
  { {65, 110, 90, 190},   /* [1] */
    Button {enabled,"Rats!"},
    {10, 62, 42, 290},    /* [2] */
    StaticText {disabled,"Are you SURE you want to  remove
this "
      "link? This cannot be undone."}
  }
};

resource 'ALRT' (260, Purgeable) {
  {100, 100, 250, 400},
  260,
  { OK, visible, sound1,/* [1] */
    OK, visible, sound1,/* [2] */
    OK, visible, sound1,/* [3] */
    OK, visible, sound1   /* [4] */
  }
};

resource 'DITL' (260, Purgeable) {
  { {80, 15, 105, 95},    /* [1] */
    Button {enabled,"Save"},
    {115, 15, 140, 95},   /* [2] */
    Button {enabled,"Cancel"},
    {116, 205, 141, 285}, /* [3] */
    Button {enabled,"No"},
    {15, 75, 63, 285},    /* [4] */
    StaticText {disabled,"Do you want to save the changes made
to ^0?"}
  }
};

resource 'ALRT' (261,Purgeable) {
  {100, 100, 200, 400},
  261,
  { OK, visible, sound1,/* [1] */
    OK, visible, sound1,/* [2] */
    OK, visible, sound1,/* [3] */
    OK, visible, sound1   /* [4] */
  }
};

resource 'DITL' (261,Purgeable) {
  { {70, 110, 95, 190},   /* [1] */
    Button {enabled,"OK"},
    {10, 62, 58, 290},    /* [2] */
    StaticText {disabled,"Doc_ID = ^0 hatCheck = ^1"}
  }
};
```

```
/* - Miscellaneous Resources - */

resource 'SIZE' (-1) {
    dontSaveScreen,
    acceptSuspendResumeEvents,
    disableOptionSwitch,
    canBackground,
    multiFinderAware,
    98304,              /* preferrred size = 128K - 32k */
    98304               /* min size = 128K - 32k */
};

type 'IAC1' as 'STR ';

resource 'IAC1' (0) {
  "Sample IAC Application - Version 1.0β1"
};

resource 'ICN#' (128, Purgeable) {
  { /* array: 2 elements */
    /* [1] */
    $"0001 0400 0002 8A00 0004 5100 0008 2080"
    $"0010 1140 0022 08A0 0044 BFD0 0089 C288"
    $"0101 C104 02A4 9082 054C 2041 080E 4020"
    $"113C 8010 2249 0008 44A0 3F04 8124 4082"
    $"4248 8041 2093 3022 1123 C814 084E 7F8F"
    $"0412 B007 0221 8007 0100 A007 0080 6007"
    $"0040 1FE7 0020 021F 0010 0417 0008 0820"
    $"0004 1040 0002 2880 0001 4500 0000 8200",
    /* [2] */
    $"0001 0400 0003 8E00 0007 DF00 000F FF80"
    $"001F FFC0 003F FFE0 007F FFF0 00FF FFF8"
    $"01FF FFFC 03FF FFFE 07FF FFFF 0FFF FFFF"
    $"1FFF FFFF 3FFF FFFF 7FFF FFFF FFFF FFFF"
    $"7FFF FFFF 3FFF FFFF 1FFF FFFF 0FFF FFFF"
    $"07FF FFFF 03FF FFFF 01FF FFFF 00FF FFFF"
    $"007F FFFF 003F FFFF 001F FFF7 000F FFE0"
    $"0007 FFC0 0003 EF80 0001 C700 0000 8200"
  }
};

resource 'ICN#' (129, purgeable) {
  { /* array: 2 elements */
    /* [1] */
    $"0FFF FC00 0800 0600 0800 05C0 0800 04A0"
    $"08F8 0450 0800 0428 081F E7F4 0800 0012"
    $"081F E01F 0800 0011 0803 FC11 0800 001D"
    $"081F 0011 0800 0011 081F FE11 0800 001D"
    $"08FF E011 0800 0011 083C 0019 0800 0011"
    $"080F C011 0800 001D 080F F811 0800 0011"
    $"08FC 0019 0800 0011 083F E011 0800 001D"
    $"0807 FE11 0800 0011 0800 0011 0FFF FFF1",
    /* [2] */
    $"0FFF FC00 0FFF FE00 0FFF FFC0 0FFF FFE0"
    $"0FFF FFF0 0FFF FFF8 0FFF FFFC 0FFF FFFE"
    $"0FFF FFFF 0FFF FFFF 0FFF FFFF 0FFF FFFF"
    $"0FFF FFFF 0FFF FFFF 0FFF FFFF 0FFF FFFF"
    $"0FFF FFFF 0FFF FFFF 0FFF FFFF 0FFF FFFF"
    $"0FFF FFFF 0FFF FFFF 0FFF FFFF 0FFF FFFF"
    $"0FFF FFFF 0FFF FFFF 0FFF FFFF 0FFF FFFF"
    $"0FFF FFFF 0FFF FFFF 0FFF FFFF 0FFF FFFF"
  }
};

resource 'BNDL' (128) {
  'IAC1',
  0,
  {
    'ICN#',
    {
      0, 128;
      1, 129
    },
    'FREF',
```

```
    (
      0, 128;
      1, 129
    )
  }
};

resource 'FREF' (128, purgeable) {
  'APPL',
  0,
  ""
};

resource 'FREF' (129, purgeable) {
  'TEXT',
  1,
  ""
};

include $$Shell("hlxEtc") "editor.code";  /* make sure we get
code segments in! */
```

Listing: Boxes.c

```
/****
 *
 *These are the routine required to handle the major dialog
 *boxes.
 ***/

# include  <types.h>
# include  <memory.h>
# include  <quickdraw.h>
# include  <toolutils.h>
# include  <windows.h>
# include  <controls.h>
# include  <dialogs.h>
# include  <textedit.h>
# include  <string.h>
# include  <resources.h>
# include  <fonts.h>
# include  <menus.h>
# include  <Events.h>

# include  <iac.h>
# define PUBLIC extern
# include  <Editor.h>

/**
 * Routine: about_box
 *
 *  This handles the about box; the current name of this app
is displayed to make working with multiple copies easier. No
other fancy effects this time... */

# define __SEG__ Main
void about_box()
{
DialogPtr    d_Ptr;
short      item_hit;
Ptr        curApNmP;     /* LOW MEMORY GLOBAL! */

  curApNmP = (Ptr) 0x910;
  PARAMTEXT (curApNmP,"","","");  /* Pascal interface ver. */
  d_Ptr = GetNewDialog (ABOUT_DLOG, nil, (WindowPtr) -1);
  ModalDialog (nil, &item_hit);
  DisposDialog (d_Ptr);
}

/**
 * Routine: linkInfo_box()
 *
 *  This routine displays a dialog box with information about
each link in the current document. */

# define SCROLL_BAR_ITEM 3
# define LIST_BOX   4

# define __SEG__ Main
void linkInfo_box()

{
short      iac_code;       /* result from IAC call */
DialogPtr  d_Ptr;
short      i, lines_vis;   /* scratch */
win_dataH  the_data_H;     /* data associated with a window */
extentH    the_extH;       /* handle to extent block */
exTable    ext_recs;       /* ptr to extent-array */
TEHandle   TextH;          /* The TextEdit handle */
ControlHandle scrollH;     /* Scroll bar handle */
String(16) ext_no;         /* holds converted values */
short      i_type;         /* for GetDItem */
Handle     i_hdnl;
Rect       i_box;
char       *msg;
short      e_cnt;          /* count of active extents */
info_tbl   census_info;    /* census info for every extent */
Point      mouse_loc;
Boolean    b;

short      item_hit = 0;

  /* get necessary handles, etc. set up */
  the_data_H = (win_dataH) GetWRefCon (myWindow);
  the_extH =  (**the_data_H).the_extents;

  iac_code = iac_census(&e_cnt, &census_info); /* que pasa,
driver? */
  if (e_cnt==0)
  {
    return;
  }

  /* set up text area */
  d_Ptr = GetNewDialog (ABOUT_LINKS, nil, (WindowPtr) -1);
  SetPort ((GrafPtr) d_Ptr);
  TextFont (monaco);     /* mono-spaced for easy layout */
  TextSize (9);
  GetDItem (d_Ptr, LIST_BOX, &i_type, &i_hdnl, &i_box);
/* text-area box */
  i_box.right += 1;        /* overlap scroll bar properly */
  FrameRect (&i_box);
  InsetRect (&i_box, 2, 2);         /* margin for drawing */
  TextH = TENew (&i_box, &i_box);
  lines_vis = (i_box.bottom - i_box.top) / 12; /* text lines
visible */

  /* Highlight button */
  GetDItem (d_Ptr, ok, &i_type, &i_hdnl, &i_box);/* button */
  InsetRect (&i_box, -4, -4);
  PenSize (3, 3);
  FrameRoundRect (&i_box, 16, 16);
  PenNormal();

  /* set up scroll bar */
  scrollH = GetNewControl (ABOUT_LINKS, d_Ptr);
  if (e_cnt > lines_vis)/* enable control */
  {
    SetCtlMax (scrollH, e_cnt);
  }

  /* put in header */
  msg = "Ext# -Doc_ID-  Hat_check  Edition\n\n";
  TEInsert (msg, (long) strlen(msg), TextH);

  /* do each extent in turn */
  for (i=0; i<e_cnt; i++)
  {
```


© The Definitive MacTutor, Vol. 4

```
    sprintf(&ext_no, "%4d", i);
    TEInsert (&ext_no, 4L, TextH);
    sprintf(&ext_no, " $%8X",census_info.ext_entry[i].doc_ID);
    TEInsert (&ext_no, 10L, TextH);
    sprintf(&ext_no, "%9d",
census_info.ext_entry[i].hat_check);
    TEInsert (&ext_no, 9L, TextH);
    sprintf(&ext_no, "%9d",census_info.ext_entry[i].ed_level);
    TEInsert (&ext_no, 9L, TextH);
    TEKey(0x0D, TextH);   /* \n to end line */
  }
  TextFont(systemFont);   /* so static text looks right */
  TextSize (12);

  /*
   *The normal modal dialog loop. You can't do anything except
look at the range of each extent. */
  while (item_hit != ok)
  {
    ModalDialog (nil, &item_hit);
    switch (item_hit)
    {
      case SCROLL_BAR_ITEM:
        GetMouse (&mouse_loc);
        GlobalToLocal (&mouse_loc);
        /* scrolling code here later */
        break;

      default:
        break;
    }
  }
  DisposeControl (scrollH);  /* clean up after ourselves */
  DisposDialog (d_Ptr);
  SetPort(FrontWindow());
}

/**
 * Routine: killLink_box
 *
 *  This is the user interface that allows a user to "cut" a
hot link.  */

# define __SEG__ Main
voidkillLink_box()
{
short      item_hit;

extern voidkill_extent();

  if (ext_active)
  {
    item_hit = StopAlert (KILL_EXT, nil); /* Are you SURE? */
    if (item_hit == ok)
    {
      kill_extent();
    }
  }
  else
  {
    item_hit = StopAlert (NOT_IN_EXT, nil); /* not in any
extent */
  }
}
```

Listing: Doc_rtns.c

```
/****
 *
 *These are the routines required to actually carry out the
document-handling functions. They are invoked from menu_tree()
and are arranged in alphabetic order to simplify finding them.
Each ensures that the menus are properlyenabled and disabled.
 */

# include  <types.h>
# include  <memory.h>
# include  <packages.h>
# include  <quickdraw.h>
# include  <toolutils.h>
# include  <windows.h>
# include  <dialogs.h>
# include  <menus.h>
# include  <textedit.h>
# include  <string.h>
# include  <files.h>
# include  <resources.h>

# include  <iac.h>
# define PUBLIC extern
# include  <Editor.h>

#define noErr    0    /* 0 for success */
#define fnfErr     -43  /* File not found */

/**
 * Routine: close_doc
 *
 *  This is the code required to close (and save if necessary)
a document. */

# define __SEG__ Main
short  close_doc()
{
win_dataH  the_data_H;     /* data associated with a window */
extentH    the_extH;       /* handle to extent block */
exTable    ext_recs;       /* ptr to array of extents */
TEHandle   TextH;          /* The TextEdit handle */
short      i, item, nm_len;
short      iac_code;       /* result from IAC call */
Str255     st0;

extern short   save_doc();  /* saved updated document */

extern WindowPtr  myWindow;     /* the text window */

# define no3 /* button id */

  the_data_H = (win_dataH) GetWRefCon (myWindow);
  the_extH =  (**the_data_H).the_extents;
  TextH =     (**the_data_H).wind_TEH;
  ext_recs =  *the_extH;

  /* check about saving, etc. */
  if ((**the_data_H).dirty)
  {
    nm_len = (**the_data_H).doc_file_nm.length;
    if (nm_len)
    {
      ParamText (&(**the_data_H).doc_file_nm, "", "", "");
    }
    else
    {
      ParamText ("Untitled", "", "", "");
    }

    item = StopAlert (SAVE_CHANGES, nil);
    switch (item)
    {
      case ok:
        save_doc();
        break;

      case cancel:
        return 0;  /* don't delete extents, etc. */
        break;

      case no:
```

```
        break;
    }
  }

  /* shut down extents... */
  for (i=0; i<(**the_data_H).ext_cnt; i++)
  {
    iac_code = iac_remove_dependency(ext_recs[i].src_doc,
                    (**the_data_H).the_slot,
                    ext_recs[i].hat_check);

    if (iac_code != noErr && iac_code != NO_SUCH_DEP)
    {
      NumToString ((long)iac_code, &st0);
      ParamText (&st0,"Remove Dependency",nil,nil);
      item = StopAlert (IAC_ERR_ALRT, nil);
    }
  }

  /* clean up storage associated with window */
  the_data_H = (win_dataH) GetWRefCon(myWindow);
  DisposHandle((Handle) (**the_data_H).the_extents);
  DisposHandle((Handle) the_data_H);
  CloseWindow (myWindow);

  /* put menu items into proper states */
  EnableItem (MyMenus[fileMenu], newCommand);
  EnableItem (MyMenus[fileMenu], openCommand);
  DisableItem (MyMenus[fileMenu], closeCommand);
  DisableItem (MyMenus[fileMenu], saveCommand);
  DisableItem (MyMenus[editMenu], 0);   /* nothing to edit */
  DisableItem (MyMenus[optionsMenu], 0);
  DisableItem (MyMenus[fontMenu], 0);
  DisableItem (MyMenus[sizeMenu], 0);

  DeleteMenu (linkdDispMenu);         /* no doc, no menu… */
  DisposeMenu (MyMenus[linkdDispMenu]);
  MyMenus[linkdDispMenu] = nil;

  DrawMenuBar;
}

/**
 * Routine: create_doc
 *
 *  This is the code for creating a new document. */

# define __SEG__ Main
short  create_doc()
{
win_dataH    the_data_H;  /* data associated with a window */

extern WindowPtr  myWindow;       /* the text window */
extern void    setup_wind();     /* set up window */

  setup_wind();           /* create window & data structures
*/
  the_data_H = (win_dataH) GetWRefCon (myWindow);
  (**the_data_H).doc_file_nm.length = 0;/* no name yet */

  /* put menu items into proper states */
  DisableItem (MyMenus[fileMenu], newCommand);
  DisableItem (MyMenus[fileMenu], openCommand);
  EnableItem (MyMenus[fileMenu], closeCommand);
  EnableItem (MyMenus[fileMenu], saveCommand);
  EnableItem (MyMenus[editMenu], 0);    /* stuff to edit */
  EnableItem (MyMenus[optionsMenu], 0);
  EnableItem (MyMenus[fontMenu], 0);
  EnableItem (MyMenus[sizeMenu], 0);
  DrawMenuBar;
}

/**
 * Routine: open_doc
 *
 *  This is the code required to open a document. After
opening the doc, reading in the text, and reloading the
extents, it checks to see if ther are any documents open of
which it was a target in a previous life, and re-establishes
those links. After that, it re-starts any links for which it
was the source. */

# define __SEG__ Main
short  open_doc()
{
IOParam    the_blk;
Handle     txtH;        /* Handle to text itself */
Handle     extRH;       /* extent-resource handle */
long int   txt_size;    /* for TextEdit */
SFReply    a_reply;     /* for SFGetFile */
Point      where;
SFTypeList the_types;
OSErr      an_err;
short      res_refNum;     /* for resource work */
short      slot_ID, vers, the_ed;   /* for IAC */
short      c_ndx, e_ndx, e_cnt;/* census-walking loops */
long       t_doc;   /* temporaries to avoid memory problems */
short      t_slot, t_hatchk;
exTable    ext_recs;       /* ptr to extent-array */
win_dataH  the_data_H;     /* data associated with a window */
win_dataH  winRH;       /* window data from saved file */
TEHandle   TextH;       /* The TextEdit handle */
info_tbl   the_census;  /* info for each dependency */

short      iac_code = noErr; /* result from IAC call */
short      doc_count = 0;    /* how many are open? */
short      ext_so_far = 0;   /* counter for 'Links' menu */

extern void  setup_wind();  /* set up window */
extern void  add_display_cmd();  /* update 'Links' menu */


  SetPt (&where, 80, 100);
  the_types[0] = (OSType) 0x54455854;   /* 'TEXT' */
  SFGetFile (&where, "", nil, 1, &the_types[0], nil,
&a_reply.good);
  if (a_reply.good)
  {
    /* an_err = SetVol(nil, &a_reply.vRefNum); /* set default
volume */
    an_err = FSOPEN (&a_reply.fName, a_reply.vRefNum, &fRef);

    the_blk.ioCompletion = nil;
    the_blk.ioRefNum = fRef;
    an_err = PBGetEOF(&the_blk.qLink, false);
    txt_size = (long int) the_blk.ioMisc;
    txtH = NewHandle (txt_size);

    HLock(txtH);
    the_blk.ioCompletion = nil;      /* set up driver
parameter block */
    the_blk.ioRefNum =     fRef;
    the_blk.ioBuffer =     *txtH;
    the_blk.ioReqCount =   txt_size;
    the_blk.ioPosMode =    fsFromStart;
    the_blk.ioPosOffset =  0;
    an_err = PBRead(&the_blk.qLink, false);

    setup_wind();   /* set up window and TE record */
    SETWTITLE (myWindow, &a_reply.fName); /*window title OK*/
    the_data_H = (win_dataH) GetWRefCon (myWindow);
    TextH = (**the_data_H).wind_TEH;
    TESetText (*txtH, txt_size, TextH);   /* TE record */
    HUnlock(txtH);
    DisposHandle (txtH);         /* no longer needed */

    /* get saved data & attach it to the window */
    res_refNum = OPENRESFILE(&a_reply.fName);
```

```
    extRH = GetResource ('EXTN', 0); /* get extents */
    DetachResource (extRH); /* remove from resource manager */
    DisposHandle((Handle) (**the_data_H).the_extents);
        /* kill old table */
    (**the_data_H).the_extents = extRH;

    winRH = (win_dataH) GetResource('EXTN', 1); /* get saved
window data */
    (**the_data_H).doc_ID =   (**winRH).doc_ID;
    (**the_data_H).ext_cnt =  (**winRH).ext_cnt;
    DisposHandle((Handle) winRH);/* no longer needed */

    the_blk.ioCompletion = nil;
    the_blk.ioRefNum =     fRef;
    an_err = PBClose(&the_blk.qLink, false);/* close file */

  /*We get a census of existing dependencies, and check them
    against our extent data. For each match, we make that the
    "available dependency" and re-establish the link. */

    iac_code = iac_census(&doc_count, &the_census);
    if (iac_code == noErr)
    {
      e_cnt = (**the_data_H).ext_cnt;
      HLock(extRH); /* IAC calls may move memory */

      for (c_ndx=0; c_ndx<doc_count; c_ndx++)/* walk census */
      {
        ext_recs = *(extentH) extRH; /* base of table */
        for (e_ndx=0; e_ndx<e_cnt; e_ndx++)/* walk extents */
        {
          if ((the_census.ext_entry[c_ndx].doc_ID ==
               ext_recs[e_ndx].src_doc) &&
              (the_census.ext_entry[c_ndx].hat_check ==
                ext_recs[e_ndx].hat_check))
          {
        /* make this dependency "available" for completion */
            iac_code = iac_available_dependency(
ext_recs[e_ndx].src_doc,ext_recs[e_ndx].hat_check);

            /* complete "available dependency" */
            t_doc = 0;      /* temps across IAC calls */
            t_hatchk = 0;   /* already made extent avail */
            t_slot = (**the_data_H).the_slot;
            iac_code = iac_complete_dependency(&t_doc,
                        &t_slot, &t_hatchk);
          (**the_data_H).the_slot = t_slot;/* could change */

          /* read initial copy of data from restarted link */
            the_ed = 0;
            iac_code = ext_read(myWindow, &the_ed, e_ndx);

            ext_so_far += 1;
            add_display_cmd(e_ndx, ext_so_far);
            /* add link to menu */
          }
        } /* for */
      } /* got census */

      if (e_cnt) /* restart links for which we are source */
      {
        ext_recs = *(extentH) extRH;
        t_doc =  (**the_data_H).doc_ID;
        t_slot = (**the_data_H).the_slot;
        for (e_ndx=0; e_ndx<e_cnt; e_ndx++)
        {
          if (ext_recs[e_ndx].src_doc ==
(**the_data_H).doc_ID)
          {
            t_hatchk = ext_recs[e_ndx].hat_check;
            the_ed = 0;
            iac_code = iac_add_dependency(&t_doc,
                    &t_slot,&t_hatchk,&the_ed);
          (**the_data_H).the_slot = t_slot;/* could change */

            /* write data to restarted dependency */
            iac_code = ext_write(TextH,
                        ext_recs[e_ndx].ext_strt,
                        /* ext_recs[e_ndx].ext_strt -
ext_recs[e_ndx].ext_end, */
                        1,    /* temporary splint */
                        t_doc,
                        t_hatchk,
                        &the_ed);

            ext_recs[e_ndx].ed_level = the_ed;
            ext_so_far += 1;
            add_display_cmd(e_ndx, ext_so_far);
            /* add link to menu */
          }
        }
      }

      HUnlock(extRH);   /* unclutter memory */
    }

    /* put menu items into proper states */
    DisableItem (MyMenus[fileMenu], newCommand);
    DisableItem (MyMenus[fileMenu], openCommand);
    EnableItem (MyMenus[fileMenu], closeCommand);
    DisableItem (MyMenus[fileMenu], saveCommand);
    EnableItem (MyMenus[editMenu], 0);    /* stuff to edit */
    EnableItem (MyMenus[optionsMenu], 0);
    EnableItem (MyMenus[fontMenu], 0);
    EnableItem (MyMenus[sizeMenu], 0);
    DrawMenuBar;
  }
}

/**
 * Routine: save_doc()
 *
 *  This is the code required to save a document.
 */

# define __SEG__ Main
short  save_doc()

{
win_dataH  the_data_H;     /* data associated with a window */
TEHandle   TextH;          /* The TextEdit handle */
IOParam    the_blk;
Handle     txtH;         /* Handle to text itself */
Handle     tmp_H;        /* temporary for resource work */
short      nm_len;
short      refNum, res_refNum;
Str255     prompt,orig_Name;
SFReply    aReply;
Point      where;
OSErr      anErr;
long int   txt_size;

  the_data_H = (win_dataH) GetWRefCon (myWindow);
  TextH =     (**the_data_H).wind_TEH;
  nm_len =  (**the_data_H).doc_file_nm.length;

  if (nm_len==0) /* invoke SFPutFile for file name */
  {
    where.h = 80;
    where.v = 80;
    SFPutFile (&where, "Name your new document", "IAC Doc",
nil, &aReply);
    if (aReply.good)
    {
      strcpy(&(**the_data_H).doc_file_nm, &aReply.fName);
/* CHECK! */
    }
    else
```

```
    {
      return 0;
    }
  }

  anErr = FSOPEN(&(**the_data_H).doc_file_nm, aReply.vRefNum,
&refNum);
  if (anErr) /* may not exist yet.. */
  {
    if (anErr==fnfErr)
    {
      anErr = CREATE(&(**the_data_H).doc_file_nm,
aReply.vRefNum, 'IAC1' , 'TEXT');
      anErr = FSOPEN(&(**the_data_H).doc_file_nm,
aReply.vRefNum, &refNum);
      (void) CREATERESFILE(&(**the_data_H).doc_file_nm);
    }
    else
    {
      return (anErr);
    }
  }
  res_refNum = OPENRESFILE(&(**the_data_H).doc_file_nm);

  /* save text itself in data fork */
  txtH = (**TextH).hText;
  txt_size = GetHandleSize (txtH);
  HLock(txtH);
  the_blk.ioCompletion =nil;
  the_blk.ioRefNum =    refNum;
  the_blk.ioBuffer =    *txtH;
  the_blk.ioReqCount =txt_size;
  the_blk.ioPosMode = fsFromStart;
  the_blk.ioPosOffset = 0;
  anErr = PBWrite(&the_blk.qLink, false);
  HUnlock(txtH);

  /* save extents in 'EXTN' resource */
  if (CountResources ('EXTN')) /* updating existing copy */
  {
    tmp_H = GetResource ('EXTN', 0); /* read old version */
    RmveResource (tmp_H);        /* kill it */
    tmp_H = GetResource ('EXTN', 1); /* read old version */
    RmveResource (tmp_H);        /* kill it */
  }
  AddResource ((**the_data_H).the_extents, 'EXTN', 0, nil);
  AddResource ((Handle) the_data_H, 'EXTN', 1, nil);
  UpdateResFile (res_refNum);

  /* close up shop till next time */
  the_blk.ioCompletion =nil;
  the_blk.ioRefNum =    refNum;
  anErr = PBClose(&the_blk.qLink, false);   /* close file */

  the_blk.ioCompletion =nil;
  the_blk.ioNamePtr =   nil;
  the_blk.ioVRefNum =   aReply.vRefNum;
  anErr = PBFlushVol (&the_blk.qLink, false);
  /* ensure disk updated */

  SETWTITLE (myWindow, &aReply.fName);

  (**the_data_H).dirty = false;
  return (0);
}

/**
 * Routine: setup_wind
 *
 *  This allocates the window and creates the TextEdit data
structure and auxiliary data record required. */

# define __SEG__ Main
void setup_wind()
{
Rect       txRect;
extentH    temp_extH;        /* handle to extents block */
Str255     tStr;

extern WindowPtr    myWindow;       /* the text window */
extern WindowRecord wRecord;
extern short       the_fNum, the_size; /* text attributes */
extern Style       the_style;

  myWindow = GetNewWindow(windowID, &wRecord, (WindowPtr) -1);
  SetPort(myWindow);
  txRect = myWindow->portRect;
  InsetRect(&txRect, 4, 0);
  TextH = TENew(&txRect, &txRect); /* Not growable, so
                                      destRect == viewRect */
  curr_ext_no = -1; /* no 'current extent' yet */

  the_data_H = (win_dataH) NewHandle(sizeof(win_data));
/* space for data */
  SetWRefCon (myWindow, (long int) the_data_H);
  /* keep with window */
  (**the_data_H).the_slot = 0; /* slot_ID for this document */
  (**the_data_H).doc_ID =   0; /* document ID for this doc */
  (**the_data_H).ext_cnt =  0; /* No. of extents with this doc
*/
  (**the_data_H).wind_TEH = TextH; /* TE handle for this
window */
  (**the_data_H).dirty =  false;
  (**the_data_H).relevent = 0;

  /* empty block of extent records */
  temp_extH = (extentH) NewHandle(sizeof(extent));
  (**temp_extH).hat_check = 0;
  (**temp_extH).ed_level =  0;
  (**temp_extH).ext_strt =  0;   /* start & end of extent
range */
  (**temp_extH).ext_end =   0;
  (**the_data_H).the_extents = temp_extH;

  (**TextH).txFont = the_fNum;   /* set text attributes */
  (**TextH).txSize = the_size;
  (**TextH).txFace = the_style;
}
```

Listing: Edit_IAC_rtns.c

```
# include  <types.h>
# include  <memory.h>
# include  <quickdraw.h>
# include  <toolutils.h>
# include  <windows.h>
# include  <dialogs.h>
# include  <menus.h>
# include  <textedit.h>
# include  <string.h>
# include  <files.h>
# include  <resources.h>

# include  <iac.h>
# define PUBLIC extern
# include  <Editor.h>

#define noErr   0   /* 0 for success */

/**
 *Routine: Set Current Extent
 *
 *  This is a local routine to set the fields in the 'current
extent' structure in the window data record. It is called by
both hotCopy and hotPaste routines, since by definitions the
extents they define become the 'current extent'. */

# define __SEG__ Main
```

```
void set_curr_ext(the_data_H, which)
  win_dataH  the_data_H;/* data associated with the window */
  short    which;     /* # of extent which is now current */
{
extentH      the_extH;       /* handle to extent block */
exTable      ext_recs;

  the_extH = (**the_data_H).the_extents;
  ext_recs = *the_extH;

  curr_ext.src_doc =  ext_recs[which].src_doc;
  curr_ext.hat_check = ext_recs[which].hat_check;
  curr_ext.ed_level = ext_recs[which].ed_level;
  curr_ext.ext_strt = ext_recs[which].ext_strt;
  curr_ext.ext_end =  ext_recs[which].ext_end;
}

/**
 *Routine: do_hotCopy
 *
 *  This routine is responsible for creating a new dependency
source and notifying the IAC driver. It is called primarily by
the "source" program.  */

# define __SEG__ Main
void do_hotCopy()

{
win_dataH  the_data_H;  /* data associated with a window */
TEHandle   TextH;       /* The TextEdit handle */
extentH    the_extH;    /* handle to extent block */
exTable    ext_recs;
long     the_doc;     /* local copies due to memory mashing */
short    slot_ID, h_check, the_ed;
short    e_cnt;
short    item_hit;       /* error processing */
Str255   err_str, str2;

short      iac_err = noErr;
Boolean    in_extent = false;

extern Boolean  chk_extent();  /* sets "current extent" if
found */
extern void add_display_cmd();  /* update 'Links' menu */
extern void set_curr_ext();   /* update global structure */

extern Boolean  ext_active;

  the_data_H = (win_dataH) GetWRefCon (myWindow);
  the_extH =  (**the_data_H).the_extents;
  TextH =     (**the_data_H).wind_TEH;
  the_doc =   (**the_data_H).doc_ID;
  slot_ID =   (**the_data_H).the_slot;
  e_cnt =     (**the_data_H).ext_cnt;
  h_check =   0;

  if (!chk_extent(TextH, the_extH, (**the_data_H).ext_cnt))
  {
    iac_err = iac_add_dependency(&the_doc, &slot_ID, &h_check,
&the_ed);
    if (iac_err == noErr)
    {
      SetHandleSize ((Handle)the_extH, sizeof(extent) *
(e_cnt+1));
      ext_recs = *the_extH;
      ext_recs[e_cnt].hat_check = h_check;
      ext_recs[e_cnt].ed_level =  the_ed;
      ext_recs[e_cnt].ext_strt =  (**TextH).selStart;
      ext_recs[e_cnt].ext_end =  (**TextH).selEnd;
      ext_recs[e_cnt].src_doc =   the_doc;
      set_curr_ext(the_data_H,e_cnt);

      curr_ext_no = e_cnt;
      e_cnt += 1;
      (**the_data_H).doc_ID =   the_doc; /* update window */
      (**the_data_H).the_slot = slot_ID;
      (**the_data_H).ext_cnt =  e_cnt;
      (**the_data_H).dirty =  true;    /* so save extents */
      ext_active = true;

      iac_err = ext_write(TextH,
                (**TextH).selStart,
                ((**TextH).selEnd - (**TextH).selStart),
                the_doc,
                h_check,
                &the_ed);

      add_display_cmd(curr_ext_no, e_cnt);  /* update menu */
    }
    else /* IAC alert here */
    {
      NumToString ((long)iac_err, &err_str);
      ParamText (&err_str, "Hot Copy", nil, nil);
      item_hit = StopAlert(IAC_ERR_ALRT, nil);
    }
  }
  else /* already-in-extent alert */
  {
    ParamText ("\"already-in-extent\"", "Hot Copy", nil, nil);
    item_hit = StopAlert(IAC_ERR_ALRT, nil);
  }
}

/**
 * Routine: do_hotPaste
 *
 *  This routine is responsible for completing a new depend-
ency and notifying the IAC driver. It is called primarily by
the "target" program. The dependencies are stored in the array
in ascending sequence to makeupdating simpler. See the header
for poll_iac() for details. */

# define __SEG__ Main
void do_hotPaste()

{
win_dataH  the_data_H;  /* data associated with a window */
TEHandle TextH;        /* The TextEdit handle */
extentH  the_extH;     /* handle to extent block */
exTable  ext_recs;
long     the_doc;      /* local copies due to memory mashing */
short    slot_ID, h_check;
short    e_cnt, startt;
short    the_ed;
short    item_hit;       /* error processing */
Str255   err_str, str2;

short      i = 0;
short      iac_err = noErr;
Boolean    in_extent = false;

extern Boolean  chk_extent();  /* sets "current extent" */
extern void add_display_cmd();  /* update 'Links' menu */

extern Boolean  ext_active;

  the_data_H = (win_dataH) GetWRefCon (myWindow);
  the_extH =  (**the_data_H).the_extents;
  TextH =     (**the_data_H).wind_TEH;
  slot_ID =   (**the_data_H).the_slot;
  e_cnt =     (**the_data_H).ext_cnt;
  the_doc =   0;    /* just link to "available" */
  h_check =   0;

  if (!chk_extent(TextH, the_extH, e_cnt))
  {
    iac_err = iac_complete_dependency(&the_doc, &slot_ID,
&h_check);
```

```
    if (iac_err == noErr) /* save "location" in TERec as
extent */
    {
      (**the_data_H).the_slot = slot_ID;  /* update window */
      ext_active = true;
      startt = (**TextH).selStart;
      SetHandleSize ((Handle)the_extH, sizeof(extent) *
(e_cnt+1));

      /* walk extent table to find entry to insert BEFORE */
      ext_recs = *the_extH;
      while (i<e_cnt)
      {
        if ( (ext_recs[i].ext_strt==0) /* at end of table */
          || (ext_recs[i].ext_strt>startt) ) /* table entry
bigger */
        {
          /* move tail of table to make space */
          BlockMove(&ext_recs[i],
                &ext_recs[i+1],
                (e_cnt - i) * sizeof(extent));
          break;
        }
        else
        {
          i += 1;
        }
      }
      ext_recs[i].hat_check = h_check;/* fill table entry */
      ext_recs[i].ext_strt =  (**TextH).selStart;
      ext_recs[i].ext_end  =  (**TextH).selEnd;
      ext_recs[i].src_doc =   the_doc;
      set_curr_ext(the_data_H,i);

      /* now read data from newly connected link */
      the_ed = 0;
      iac_err = ext_read(myWindow, &the_ed, i);

      extent_count += 1;          /* global count */
      curr_ext_no = i;
      (**the_data_H).ext_cnt = e_cnt + 1;
      (**the_data_H).dirty = true;
      add_display_cmd(i, extent_count);  /* update menu */
    }
    else
    {
      NumToString ((long)iac_err, &err_str);
      ParamText (&err_str, "Hot Paste", nil, nil);
      item_hit = StopAlert(IAC_ERR_ALRT, nil);
    }
  }
  else /* already-in-extent alert */
  {
    ParamText ("\"already-in-extent\"", "Hot Paste", nil,
nil);
    item_hit = StopAlert(IAC_ERR_ALRT, nil);
  }
}
```

Listing: Edit_rtns.c

```
/****
 *
 *These are the routines needed for the actual text-editing
functions. They are arranged alphabetically for ease of
finding them. */

# include  <types.h>
# include  <errors.h>
# include  <memory.h>
# include  <packages.h>
# include  <quickdraw.h>
# include  <toolutils.h>
# include  <fonts.h>
# include  <windows.h>
# include  <dialogs.h>
# include  <menus.h>
# include  <textedit.h>
# include  <string.h>
# include  <files.h>
# include  <resources.h>

# include  <iac.h>
# define PUBLIC extern
# include  <Editor.h>

 /**
 * Routine: do_clear
 *
 *  This is the routine that carries out the "clear" editing
function. It uses the normal TE routine to manipulate the
text. In addition, it checks the source extents to see if any
of them are affected by it, and if so,it will post the changed
extent to the IAC driver for everybody else's use. */

# define __SEG__ Main
void do_clear()
{
short    this_ed;      /* edition (updated) */
short    hit_ext;      /* extent affect by this call */
short    i;            /* scratch */
short    strt, endd;   /* range in TE record */
win_dataH  the_data_H;  /* data associated with a window */
extentH    the_extH;   /* handle to extent block */
extentP    extP;
TEHandle   TextH;      /* The TextEdit handle */
Handle     ext_data;   /* data for extent */

short      iac_code = noErr; /* result from IAC call */
Boolean    in_extent = false;
Boolean    ext_killed = false;

extern Boolean  chk_extent();   /* sets "current extent" if
found */
extern void kill_extent();

  /* get necessary handles, etc. set up */
  the_data_H = (win_dataH) GetWRefCon (myWindow);
  TextH =     (**the_data_H).wind_TEH;
  the_extH =  (**the_data_H).the_extents;

  /* Check extents before actually cutting. Remember which one
and we'll update the edition after doing the delete... */
  in_extent = chk_extent(TextH,the_extH,
(**the_data_H).ext_cnt);

  TEDelete (TextH);
  (**the_data_H).dirty = true;
  /* if needed, write revised extent to IAC driver */
  if (in_extent)    /* this delete affected the extent */

  {
    /* adjust range for extent */

    /* If the entire extent was removed, tell IAC driver */
    if (ext_killed)
    {
      iac_code = iac_remove_dependency((**the_data_H).doc_ID,
                        (**the_data_H).the_slot,
                        curr_ext.hat_check);
      kill_extent(); /* always removes current extent */
    }
    else /* write to driver */
    {
      this_ed = curr_ext.ed_level;
      /* data setup here */
      iac_code = iac_write_data((**the_data_H).doc_ID,
                    curr_ext.hat_check,
                    &this_ed,
```

```
                    1,
                    ext_data);
    /* now adjust extent */
    }
  }
}

 /**
 * Routine: do_copy
 *
 *  This is the routine that carries out the "copy" editing
function. It uses the normal TE routine to manipulate the
text.  */

# define __SEG__ Main
void do_copy()

{
short       this_ed;      /* edition (updated) */
short       i;            /* scratch */
short       strt, endd;   /* range in TE record */
win_dataH   the_data_H;   /* data associated with a window */
extentH     the_extH;     /* hand to extent block */
extentP     extP;
TEHandle    TextH;        /* The TextEdit handle */
Handle      ext_data;     /* data for extent */

short       iac_code = noErr; /* result from IAC call */
Boolean     in_extent = false;

extern Boolean  chk_extent();   /* sets "current extent" */

  /* get necessary handles, etc. set up */
  the_data_H = (win_dataH) GetWRefCon (myWindow);
  TextH =      (**the_data_H).wind_TEH;
  the_extH =  (**the_data_H).the_extents;
in_extent = chk_extent(TextH,the_extH,(**the_data_H).ext_cnt);
  TECopy (TextH);
}

 /**
 * Routine: do_cut
 *
 *  This is the routine that carries out the "cut" editing
function. It uses the normal TE routine to manipulate the
text. In addition, it checks the source extents to see if any
of them are affected by it, and if so,it will post the changed
extent to the IAC driver for everybody else's use. */

# define __SEG__ Main
void do_cut()

{
short       this_ed;      /* edition (updated) */
short       hit_ext;      /* extent affect by this */
short       i;            /* scratch */
short       strt, endd;   /* range in TE record */
win_dataH   the_data_H;   /* data associated with a window */
extentH     the_extH;     /* hand to extent block */
extentP     extP;
TEHandle    TextH;        /* The TextEdit handle */
Handle      ext_data;     /* data for extent */
OSErr       an_err;

short       iac_code = noErr; /* result from IAC call */
Boolean     in_extent = false;
Boolean     ext_killed = false;

extern Boolean  chk_extent();   /* sets "current extent"  */
extern void kill_extent();

  /* get necessary handles, etc. set up */
  the_data_H = (win_dataH) GetWRefCon (myWindow);
  TextH =      (**the_data_H).wind_TEH;
  the_extH =  (**the_data_H).the_extents;

  /* Check extents before actually cutting. Remember which one
and we'll update the edition after doing the cut...  */
in_extent = chk_extent(TextH,the_extH,(**the_data_H).ext_cnt);
  TECut (TextH);
  (**the_data_H).dirty = true;

  /* if needed, write revised extent to IAC driver */
  if (in_extent)    /* this cut affected the extent */
  {
    /* adjust range for extent */

    /* If the entire extent was removed, tell IAC driver */
    if (ext_killed)
    {
      iac_code = iac_remove_dependency((**the_data_H).doc_ID,
                        (**the_data_H).the_slot,
                        curr_ext.hat_check);
      kill_extent();  /* always removes current extent */
    }
    else /* write to driver */
    {
      HLock((Handle) TextH);
      /* data setup here */
      an_err = PtrToHand (&(**TextH).selStart,&ext_data,
            (long) curr_ext.ext_end - curr_ext.ext_strt + 1);
      HUnlock((Handle) TextH);
      this_ed = curr_ext.ed_level;
      iac_code = iac_write_data((**the_data_H).doc_ID,
                    curr_ext.hat_check,
                    &this_ed,1,ext_data);
    } /* end else */
  } /* end in_extent */
}

/**
 * Routine: do_key
 *
 *  This is the routine that handles normal typing. It does
NOT notify the IAC driver after every keystroke. */

# define __SEG__ Main
void do_key(msg)
  long   msg;   /* event record message field */

{
win_dataH   the_data_H;   /* data associated with a window */
extentH     the_extH;     /* hand to extent block */
TEHandle    TextH;        /* The TextEdit handle */
char        the_ch;
short       e_cnt;

Boolean       in_extent = false;

extern Boolean  chk_extent();

#define charCodeMask 0x000000FF
/*copied from event.h to conserve space */

  /* get necessary handles, etc. set up */
  the_data_H = (win_dataH) GetWRefCon (myWindow);
  TextH =      (**the_data_H).wind_TEH;
  the_extH =  (**the_data_H).the_extents;
  the_ch =  (char) (msg & charCodeMask);
  e_cnt =      (**the_data_H).ext_cnt;

  in_extent = chk_extent(TextH, the_extH, e_cnt);
  TEKey(the_ch, TextH);
  (**the_data_H).dirty = true;

  /* now adjust extent if necessary */
  if (in_extent)
  {
```

```
    if (the_ch==BS)   /* backspace - shrink extent */
    {
      curr_ext.ext_end -= 1;
    }
    else       /* insertion - expand extent */
    {
      curr_ext.ext_end += 1;
    }
  }
}

/**
 * Routine: do_paste
 *  This is the routine that carries out the "paste" editing
function. It uses the normal TE routine to manipulate the
text. In addition, it checks the source extents to see if any
of them are affected by it, and if so,it will post the changed
extent to the IAC driver for everybody else's use. */

# define __SEG__ Main
void do_paste()
{
short      this_ed;      /* edition (updated) */
short      hit_ext;      /* extent affect by this */
short      i;            /* scratch */
short      delta;        /* change in extent size */
short      strt, endd;   /* range in TE record */
win_dataH  the_data_H;   /* data associated with a window */
extentH    the_extH;     /* hand to extent block */
TEHandle   TextH;        /* The TextEdit handle */
Handle     ext_data;     /* data for extent */

short      iac_code = noErr; /* result from IAC call */
Boolean    in_extent = false;

extern Boolean  chk_extent();   /* sets "current extent"  */

  /* get necessary handles, etc. set up */
  the_data_H = (win_dataH) GetWRefCon (myWindow);
  TextH =     (**the_data_H).wind_TEH;
  the_extH =  (**the_data_H).the_extents;

  /* Check extents before actually cutting. Remember which one
and we'll update the edition after doing the paste... */
  in_extent = chk_extent(TextH, the_extH,
(**the_data_H).ext_cnt);

  TEPaste (TextH);
  (**the_data_H).dirty = true;

  /* if needed, write revised extent to IAC driver */
  if (in_extent)    /* this paste affects the extent */
  {
    /* adjust range for extent */
    /* write to driver */
    this_ed = curr_ext.ed_level;
      /* data setup here */
    iac_code = iac_write_data((**the_data_H).doc_ID,
                  curr_ext.hat_check,
                  &this_ed,1,ext_data);

    curr_ext.ext_end += delta;   /* now update extent */
    curr_ext.ed_level = this_ed;
  }
}

/**
 * Routine: kill_extent
 *  This removes the current extent from the table and
compresses the table. The count and "active flag" are also
updated. */

# define __SEG__ Main
void kill_extent()/* always removes current extent */
{
win_dataH  the_data_H;     /* data associated with a window */
extentH    the_extH;       /* handle to extent block */
exTable    ext_recs;
long     the_doc;      /* local copies due to memory mashing */
short      slot_ID, h_check;
short      e_cnt, startt;
short      i;

short      iac_code = noErr; /* result from IAC call */

extern Boolean    ext_active;

  the_data_H = (win_dataH) GetWRefCon (myWindow);
  the_extH =  (**the_data_H).the_extents;
  e_cnt =     (**the_data_H).ext_cnt;
  startt =  curr_ext.ext_strt;

  ext_recs = *the_extH;    /* point to table base */
  for (i=0; i<e_cnt; i++)
  {
    if (ext_recs[i].ext_strt == startt) /* found current
extent */
    {
      iac_code = iac_remove_dependency((**the_data_H).doc_ID,
                      (**the_data_H).the_slot,
                      ext_recs[i].hat_check);
      BlockMove(&ext_recs[i+1],  /* move tail down */
            &ext_recs[i],
            (e_cnt-i)*sizeof(extent));
      (**the_data_H).ext_cnt -= 1;
      (**the_data_H).dirty = true;
      break;
    }
  }
  ext_active = false;
}
```

Listing: IAC.c

```
/***
 * File:   IAC.c
 * Package:  Inter Application Communications
 * Description: This is the interface package to the driver for
 *  the use of application programs.
 * Author(s):
 *FEA    6/19/88  */

# include  <types.h>
# include  <files.h>
# include  <memory.h>
# include  <osutils.h>
# include  <serial.h>
# include  <iac.h>
# include  <dialogs.h>

# define MIN_BLK_SIZE 0xC
# define DEBUG  false
# define IAC_ERR_ALRT 257

static short  iac_ref_num;

/**
 * Routine: iac_open
 *  The IAC driver is opened by this call and the ioRefNum
saved so it doesn't need to be passed with all calls. A null
value is returned if the open is successful; otherwise the
operating system error is  returned. */
short  iac_open()

{
SysEnvRec theWorld;/* ALMOST complete knowledge about world */
THz      s_ZoneP, a_ZoneP; /* so can check zone adjacency */
OSErr    the_err;
```

```
short    result = 0;

  the_err = SysEnvirons(1, &theWorld);
  if (theWorld.systemVersion < 0x0420)    /* too early! */
  {
    result = EARLY_SYS;    /* "no multifinder" */
  }
  else
  {
  /*check for multifinder active by checking if system zone is
      adjacent to application zone, which never happens under
      MultiFinder. */

    s_ZoneP = SystemZone();
    a_ZoneP = ApplicZone();
    if ( ((long) s_ZoneP->bkLim + MIN_BLK_SIZE) == (long)
a_ZoneP )
    {
      result = EARLY_SYS;    /* zones adjacent - no MFinder */
    }
    else /* now try to actually open the IAC driver */
    {
      SysBeep(1);
      result = OPENDRIVER("\p.IAC", &iac_ref_num);
    }
  }
  return(result);
}

/**
 * Routine: iac_add_dependencyz
 *  This is used to inform the IAC driver that a new depend-
ency has been established and should be added to its internal
tables. A null value is returned if the open is successful;
otherwise the operating system error is returned. */

short  iac_add_dependency(doc_id, slot_id, hat_check, edition)
  long *doc_id; /* identifies source document (permanent) */
  short  *slot_id;  /* identifies source document (session) */
  short  *hat_check;/* extent identifier */
  short  *edition;  /* how many times extent has changed
(session) */

{
IOParam    the_blk;
OSErr    the_err;

struct {
  short  func;
  long doc_id;
  short  slot_id;
  short  hat_check;
  short  edition;
} my_params;

# if DEBUG
  return(the_err=noErr);  /* short circuit for testing */
# endif

  my_params.func = 1;    /* set up private parameter block */
  my_params.doc_id = *doc_id;
  my_params.slot_id = *slot_id;
  my_params.hat_check = *hat_check;

  the_blk.ioCompletion = nil; /* set up driver param block */
  the_blk.ioRefNum = iac_ref_num;
  the_blk.ioBuffer = &my_params;
  the_blk.ioReqCount = sizeof(my_params);
  the_blk.ioPosMode = fsFromStart;
  the_blk.ioPosOffset = 0;
  the_err = PBWrite(&the_blk.qLink, false);
  /* add the dependency */

  if (the_err == noErr)    /* update output parameters */
  {
    *doc_id = my_params.doc_id;
    *slot_id = my_params.slot_id;
    *hat_check = my_params.hat_check;
    *edition = my_params.edition;
  }
  return(the_err);
}

/**
 * Routine: iac_complete_dependency
 *  This is used to inform the IAC driver of a document that
is interested in a particular dependency. A null value is
returned if the open is successful; otherwise the operating
system error is returned. */

short  iac_complete_dependency(doc_id, slot_id, hat_check)
  long *doc_id; /* identifies source document (permanent) */
  short  *slot_id;  /* identifies source document (session) */
  short  *hat_check;/* extent identifier */

{
IOParam    the_blk;
OSErr    the_err;

struct {
  short  func;
  long doc_id;
  short  slot_id;
  short  hat_check;
} my_params;

# if DEBUG
  return(the_err=noErr);  /* testing without MF */
# endif

  my_params.func = 2;    /* set up private parameter block */
  my_params.doc_id = *doc_id;
  my_params.slot_id = *slot_id;
  my_params.hat_check = *hat_check;

  the_blk.ioCompletion = nil;/* set up driver param block */
  the_blk.ioRefNum = iac_ref_num;
  the_blk.ioBuffer = &my_params;
  the_blk.ioReqCount = sizeof(my_params);
  the_blk.ioPosMode = fsFromStart;
  the_blk.ioPosOffset = 0;
  the_err = PBWrite(&the_blk.qLink, false);

  if (the_err == noErr)    /* update output parameters */
  {
    *doc_id = my_params.doc_id;
    *slot_id = my_params.slot_id;
    *hat_check = my_params.hat_check;
  }

  return(the_err);
}

/**
 * Routine: iac_remove_dependency
 *
 *  This is used to inform the IAC driver that a document is
no longer interested in a particular dependency. If the
document is the original source all "targets" will be informed
that there is no longer any such extent; if all targets lose
interest the source will be informed that it no longer needs
to update the extent. */

short  iac_remove_dependency(doc_id, slot_id, hat_check)
  long doc_id;    /* identifies source document (permanent) */
  short  slot_id; /* identifies source document (session) */
  short  hat_check; /* extent identifier */
```

```
{
IOParam    the_blk;
OSErr    the_err;

struct {
  short  func;
  long doc_id;
  short  slot_id;
  short  hat_check;
} my_params;

# if DEBUG
  return(the_err=noErr);  /* testing without MF */
# endif

  my_params.func = 3; /* set up private parameter block */
  my_params.doc_id = doc_id;
  my_params.slot_id = slot_id;
  my_params.hat_check = hat_check;

  the_blk.ioCompletion = nil; /* set up driver param block */
  the_blk.ioRefNum = iac_ref_num;
  the_blk.ioBuffer = &my_params;
  the_blk.ioReqCount = sizeof(my_params);
  the_blk.ioPosMode = fsFromStart;
  the_blk.ioPosOffset = 0;
  the_err = PBWrite(&the_blk.qLink, false);

  return(the_err);
}

/**
 * Routine: iac_available_dependency
 *
 *  This sets the indicated extent as the "available extent"
to be used as the source for future defaulted
"complete_dependency"  calls.  */

short  iac_available_dependency(doc_id, hat_check)
  long doc_id;    /* identifies source document (permanent) */
  short  hat_check; /* extent identifier */

{
IOParam    the_blk;
OSErr    the_err;

struct {
  short  func;
  long doc_id;
  short  hat_check;
} my_params;

# if DEBUG
  return(the_err=noErr);  /* for testing without MF */
# endif

  my_params.func = 4; /* set up private parameter block */
  my_params.doc_id = doc_id;
  my_params.hat_check = hat_check;

  the_blk.ioCompletion = nil;  /* set up driver param block */
  the_blk.ioRefNum = iac_ref_num;
  the_blk.ioBuffer = &my_params;
  the_blk.ioReqCount = sizeof(my_params);
  the_blk.ioPosMode = fsFromStart;
  the_blk.ioPosOffset = 0;
  the_err = PBWrite(&the_blk.qLink, false); /* this depend-
ency is now "available" */

  return(the_err);
}

/**
 * Routine: iac_status
 *
 *  This allows the application program to find out what's
going on. */

short  iac_status(slot_id, vers_id, doc_count, extent_count)
  short  slot_id;   /* identifies the inquiring document
(session) */
  short  *vers_id;    /* driver version*100 */
  short  *doc_count;  /* count of active documents */
  short  *extent_count;  /* count of extents relevant to
inquiring doc */

{
IOParam    the_blk;
OSErr    the_err;

struct {
  short  func;
  short  slot_id;
  short  vers_id;
  short  doc_count;
  short  extent_count;
} my_params;

# if DEBUG
  return(the_err=noErr);  /* for testing without MF */
# endif

  my_params.func = 5;   /* set up private parameter block */
  my_params.slot_id = slot_id;

  the_blk.ioCompletion = nil;  /* set up driver param block */
  the_blk.ioRefNum = iac_ref_num;
  the_blk.ioBuffer = &my_params;
  the_blk.ioReqCount = sizeof(my_params);
  the_blk.ioPosMode = fsFromStart;
  the_blk.ioPosOffset = 0;
  the_err = PBRead(&the_blk.qLink, false);/* read IAC status*/

  if (the_err == noErr)   /* update output parameters */
  {
    *vers_id = my_params.vers_id;
    *doc_count = my_params.doc_count;
    *extent_count = my_params.extent_count;
  }

  return(the_err);
}

/**
 * Routine: iac_census
 *
 *  This provides identifying info for all registered extents.
 */

short  iac_census(extent_count, extent_info)
  short    *extent_count;  /* count of extents registered */
  info_tblP  extent_info;  /* Ptr to table of info for each
extent */

{
IOParam    the_blk;
OSErr    the_err;
short    i;

struct {
  short    func;
  short    extent_count;
  info_rec extent_info[MAX_EXTS];
} my_params;

# if DEBUG
  return(the_err=noErr);  /* testing without MF */
```

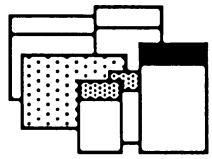
```
# endif

  my_params.func = 6; /* set up private parameter block */
  the_blk.ioCompletion = nil;  /* set up driver param block */
  the_blk.ioRefNum = iac_ref_num;
  the_blk.ioBuffer = &my_params;
  the_blk.ioReqCount = sizeof(my_params);
  the_blk.ioPosMode = fsFromStart;
  the_blk.ioPosOffset = 0;
  the_err = PBRead(&the_blk.qLink, false);/* IAC status */

  if (the_err == noErr) /* update output parameters */
  {
    *extent_count = my_params.extent_count;
    BlockMove (&my_params.extent_info[0],
           (Ptr) extent_info,
           (long) my_params.extent_count * sizeof(info_rec));
  }
  return(the_err);
}

/**
 * Routine: iac_write_data
 *  This updates the data for the specified extent, resulting
in a new change level. */

short  iac_write_data(doc_id, hat_check, edition, fmt_count,
ext_data)
  long doc_id;    /* identifies source document (permanent) */
  short  hat_check; /* extent identifier */
  short  *edition;  /* how many times extent has changed
(session) */
  short  fmt_count; /* number of formats being written */
  Handle ext_data;  /* Handle to actual data */

{
IOParam    the_blk;
OSErr    the_err;

struct {
  short  func;
  long doc_id;
  short  hat_check;
  short  edition;
  short  fmt_count;
  Handle the_dataH;
} my_params;

# if DEBUG
  return(the_err=noErr);  /* for testing without MF */
# endif

  my_params.func = 7;    /* set up private parameter block */
  my_params.doc_id = doc_id;
  my_params.hat_check = hat_check;
  my_params.fmt_count = fmt_count;
  my_params.the_dataH = ext_data;
  the_blk.ioCompletion = nil;  /* set up driver param block */
  the_blk.ioRefNum = iac_ref_num;
  the_blk.ioBuffer = &my_params;
  the_blk.ioReqCount = sizeof(my_params);
  the_blk.ioPosMode = fsFromStart;
  the_blk.ioPosOffset = 0;
  the_err = PBWrite(&the_blk.qLink, false); /* update
dependency */

  if (the_err == noErr)   /* update output parameters */
  {
    *edition = my_params.edition;
  }
  return(the_err);
}
/**
 * Routine: iac_read_data
 *  This is used to retrieve the actual data for the latest
change_levelfor the specified extent. The IAC driver will
record that the inquiring document has read the data.ext_data
will be resized by the driver to hold the data.  */

short  iac_read_data(doc_id, slot_id, hat_check, edition,
fmt_pref,fmt_code, ext_data)
  long doc_id;    /* identifies source document (permanent) */
  short  slot_id;   /* identifies the source document (ses-
sion) */
  short  hat_check;    /* extent identifier */
  short  *edition;     /*times extent has changed (session) */
  long fmt_pref[3]; /* prefd formats, descendg desirability */
  long *fmt_code;    /* format returned to caller */
  Handle ext_data;     /* Handle to actual data */
{
IOParam    the_blk;
OSErr    the_err;

struct {
  short  func;
  long doc_id;
  short  slot_id;
  short  hat_check;
  short  edition;
  long fmt_pref[3];
  long fmt_code;
  Handle ext_data;
} my_params;

# if DEBUG
  return(the_err=noErr);  /* for testing without MF */
# endif
  my_params.func = 8;    /* set up private parameter block */
  my_params.doc_id = doc_id;
  my_params.slot_id = slot_id;
  my_params.hat_check = hat_check;
  my_params.edition = *edition;
  my_params.fmt_pref[0] = fmt_pref[0];
  my_params.fmt_pref[1] = fmt_pref[1];
  my_params.fmt_pref[2] = fmt_pref[2];
  my_params.ext_data = ext_data;
  the_blk.ioCompletion = nil;  /* set up driver param block */
  the_blk.ioRefNum = iac_ref_num;
  the_blk.ioBuffer = &my_params;
  the_blk.ioReqCount = sizeof(my_params);
  the_blk.ioPosMode = fsFromStart;
  the_blk.ioPosOffset = 0;
  the_err = PBRead(&the_blk.qLink, false);/* read the data */
  if (the_err == noErr)
  {
    *edition = my_params.edition;
    *fmt_code = my_params.fmt_code;}
  return(the_err);
}
```

Listing: SysEnvs.a

```
; Glue for SysEnvirons that was skipped by Apple
; F. Alviani
; 7/88
    INCLUDE      'TRAPS.A'  ;Include Memory manager macros.
    PRINT ON
SYSENVIRONS      FUNC EXPORT
    movea.l      4(sp),a0 ;a0 contains ptr to world record
    move.w       8(sp),d0 ;d0 contains desired version
    _SysEnvirons
    movea.l      (sp)+,a0 ;get return addr, remove
    lea          6(sp),sp ;pop parameters
    move         d0,(sp)  ;put result in place
    jmp          (a0)     ; go home
    DC.B 'SYSENVIR'

    END
```

Ç Workshop

Tool Window Manager



Thomas Fruin
The Netherlands
MacTutor Vol. 4 No. 12

The Tool Window Manager

Thomas Fruin is a graduate student of Computer Science at Leiden Universit, who also does study-related work for the two universities in Amsterdam, the University of Amsterdam and the Free University. He is the president and co-founder of VAMP (Vereniging Apple Macintosh Programmeurs), the main Dutch association for Macintosh programmers. Otherwise, he has done various odd Mac-related jobs to earn money, usually writing software (educational, communications, etc.).

The Macintosh user interface has been enhanced with some interesting new features lately: tear-off menus, pop-up menus, hierarchical menus, window layers, color, and even a new name: human interface. While most of these additions are in the new ROMs and the latest System file, not everything has been dealt with by Apple. Most notably is the lack of code to support tear-off menus, the most recent novelty and pioneered by HyperCard. They are something you are going to have to code all by yourself (well, with a bit of help from others). Darryl Lovato addressed the problem of tearing off a menu first, followed by an in depth article by Don Melton and Mike Ritter. In my contribution I will concentrate on keeping windows (torn-off menus or otherwise) floating above the others on your desktop, by presenting an alternative window manager - called the Tool Window Manager - that does all the work.

While tear-off menus are new to the Mac, floating windows are not. For several years now there have been applications putting their tools, patterns and other wares in these little windows, like FullPaint and PageMaker to name just two. Tool windows are a tricky business though, and it takes some careful planning to get everything right. Even commercial programs make mistakes at this (see the sidebar on the bugs in FullPaint and MacPaint).

Tool Windows Have a Special Appearance

First of all, since tool windows always lie on top, there is no distinction between an active or inactive tool window. This means that the tool should either have no title bar (like in FullPaint), or a title bar that never changes its appearance (like in HyperCard or Claris MacPaint). The latter case requires a special window definition function. This is not the subject of this article, however, since window definition functions have already been dealt with by MacTutor. Besides, you can cheat by experimenting with the WDEF for tool windows in the new MacPaint. Just copy WDEF 34 from your copy of MacPaint with ResEdit and paste it in your own application. Of course you cannot distribute it ... (but you knew that, didn't you?)

Don Melton and Mike Ritter's comments on the user interface guidelines for tool windows, and especially their additions to the guidelines, are a good idea. Since the article came out, I have modified my manager to incorporate these additions: when a non-application window is brought to the front (usually a desk accessory), my manager will unhighlight all the tool windows. When that window disappears, all the tools are highlighted again. I also added this behavior when a modal dialog window is brought to the front, because it seems to make sense. Finally, the unhighlighting of the tool windows also takes place when another application is activated under MultiFinder. It is up to the application that uses the Tool Window Manager to hide the tool windows when the application is juggled out, since it may not always want to hide the tools (one of them may be some kind of status window, for example).

Tool Windows or GhostWindows?

Tool windows are also sometimes called ghost windows. This is probably due to the obscure GhostWindow low memory global that is mentioned in Inside Macintosh volume I, page 287. You had better forget about it. I'm not only doing this as a public service (since Apple is discouraging people to use low memory globals), but also because it really doesn't help in our case. As in almost every part of Inside Macintosh, the information is there, but it takes some careful reading to understand what the paragraph is about. Besides, GhostWindow will only let you have one tool window at a time, and we want to create as many tool windows as will fit in memory (it's ok with me if you only want two or three). So it seems the only way out is a front end for all the Window Manager routines that could cause a tool window to become obscured. This is what I have written in the Tool Window Manager, or TWindow Manager for short.

The Mad T Party

The TWindow Manager is a set of 10 functions that replace calls to functions in the Window and Event Managers in your source file. These new functions do some extra processing before they call the Window and Event Manager themselves. The names of these TWindow Manager routines are the same as their corresponding Window and Event Manager functions, except for a prefixed "T", as in TSelectWindow and TGetNextEvent. Their parameters are almost the same as the original functions, with a few slight modifications. See the explanations below for details. There are six categories of window manipulations that need to be replaced by TWindow Manager functions to work correctly with tool windows: (1) Adding windows to the desktop: use TNewWindow, TGetNewWindow and TShowWindow. (2) Removing windows: use TCloseWindow, TDisposeWindow and THideWindow. (3) Bringing lower windows to the top with

© The Definitive MacTutor, Vol. 4

TSelectWindow. (4) Dragging with TDragWindow. (5) Utility functions you will need every now and then: TFrontWindow and the new TInitWindows and TGetWKind. (6) Event handling: TGetNextEvent. This last function is included, because activate and deactivate events are handled differently when there are tool windows about. Note that I did not mention resizing windows (yes, I'm mentioning it now, thank you). Amazingly, GrowWindow is smart enough to draw its grow outline below any windows that are lying on top of it. It's always nice to have less work to do.

There are three source files that go with this article. TWindows.c is the TWindow Manager itself, written in MPW C, and can be compiled into a separate module for later linking with your own program. TWindows.h is a file for inclusion by your program. It holds common data types and the TWindow function types. Your program will need this file if it wants to call TWindow Manager functions and use constants such as toolKind and anyKind. Finally, TWindowTester.c (also in MPW C) is an example program that uses the TWindow Manager. It is a familiar 'vanilla' Macintosh program that lets you create normal and tool windows to experiment with, as may as you like (see figure 1). You can select and drag windows, and open desk accessories as well. (Note that I got the tool window definition function from MacPaint 2.0, as I described above. In the source I am distributing, the windows look more like FullPaint's tools, because MacPaint's defproc is not included.) The menus called Windows and Tools let you make any document window or tool invisible and visible again, even if it isn't in front. I added these menus myself to test the robustness of the TWindow Manager. Note that you can't do anything useful with TWindowTester (although you might derive great pleasure from opening, say, 100 windows). For example, the scroll bars in the document windows don't work. I just put them there so you can see that windows get activated and deactivated properly.

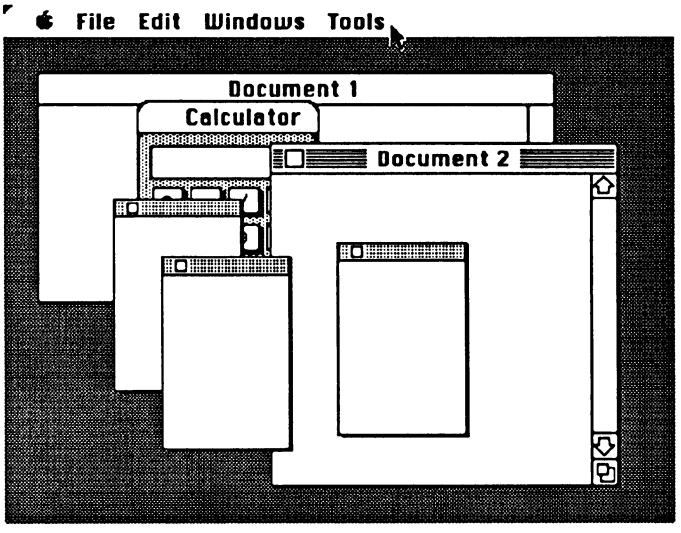
Background Information, Utility Functions

The TWindow Manager keeps tool windows and document windows apart by storing a special value in the WindowRecord's windowKind field. This value is defined as the toolKind constant (equal to 30000) in the source file for TWindows. If you like, you can modify it to a value more suitable to your application, but there should be little conflicts as is. If any of the TWindow Manager functions wants to know what kind of window it is dealing with, it calls the utility function TGetWKind. This function is precisely for determining the windowKind value of a window. Why write a function for such a simple task? Well, TGetWKind doesn't just return the window's windowKind, but does some extra processing as well. If the window is a desk accessory window (with a negative windowKind), TGetWKind will always return the constant systemKind (equal to -1). If the window is a dialog window, with windowKind equal to the toolbox constant dialogKind, TGetWKind only returns dialogKind if the window is a modal dialog window. If the dialog is modeless, TGetWKind returns the toolbox constant userKind (the toolbox constant for normal document windows). This is because modeless dialog windows behave almost exactly the same as document windows, i.e. they lie behind tool windows. TGetWKind figures out what kind of dialog it is by examining the window's variation code and definition procedure. Window definition procedures have an (optional) header embedded in their code that holds information like the WDEF's resource ID (see Technical Note 110 for more information about WDEFs - and make sure you have the "real" 110). The dialog is considered modeless if its window is a document window: this is when the WDEF's resource ID is zero and the variation code is equal to documentProc, noGrowDocProc or zoomDocProc. To play it safe, I let TGetWKind also check if the resource type in the header is really 'WDEF'. If the WDEF doesn't have a header (like the MacPaint tool window for example), it's extremely unlikely that there will be a 'WDEF' string embedded at that particular position in the code.

The Window Manager in the Macintosh ROM has (as far as I can determine) only one global variable dealing with window positions, called WindowList - at location $9D6. It holds the very first window in the window list. By comparison, my TWindow Manager has three position globals of its own: frontToolWindow, backToolWindow and frontDocWindow. FrontToolWindow points to the frontmost visible tool window, backToolWindow to the backmost visible tool window, and frontDocWindow to the frontmost visible document window. Two other global variables I use mimic the CurActivate ($A64) and CurDeactive ($A68) low memory globals in the Macintosh: toBeActivated and toBeDeactivated. They hold the real windows that need activating and deactivating, and my TGetNextEvent takes them into account when it returns activate events. They are set by various TWindow Manager routines whenever activate events need to be posted. Finally I have two utility global variables, firstRgn and secondRgn. All these seven globals need to be initialized by the TInitWindows function, before you start using any functions in the TWindow Manager.

TFrontWindow is my replacement for FrontWindow. You can specify a window type in the windowKind parameter (note that FrontWindow has no parameters at all). This window type

Figure 1: The TWindowTester Application



is one of the constants toolKind, userKind or anyKind. Usually you will want to call TFrontWindow with windowKind equal to userKind; it will then return the frontmost visible document window (the active window), ignoring any tool windows that may be lying above it (unless there is a desk accessory or modal dialog on top - it then returns the DA's or dialog's window). Calling TFrontWindow with windowKind equal to toolKind returns the frontmost visible tool window (again taking DAs and dialogs into account). When you call TFrontWindow with anyKind as the parameter, it behaves exactly like the original FrontWindow function - it simply returns the frontmost visible window.

Selecting Windows

I'm saving the subjects of adding and removing windows for later, because they are the trickiest ones to handle. TSelectWindow, by comparison, is much easier. Here, the main problem is to bring a document window to the "front" while keeping it behind any tool windows that may be visible. Apple, in its infinite wisdom, already has a function that almost does this: it's called SendBehind and it's listed on page I-286 in Inside Macintosh. You use it to move a window closer to the front (ah, so that's why it's called SendBehind), and specify the window behind which your window should be moved. In our case this will usually be the backmost tool window. Note what Inside Macintosh has to say about this: "If you're moving theWindow closer to the front (that is, if it's initially even further behind behindWindow), you must make the following calls after calling SendBehind:

```
wPeek = POINTER( theWindow );
PaintOne( wPeek, wPeek^.strucRgn );
CalcVis( wPeek );
```

This means that you have to do some low-level stuff, because SendBehind isn't really meant to bring windows forward. PaintOne (5 pages further) is a routine that whitewashes the newly exposed part of the window. CalcVis (on the next page in Inside Macintosh) recalculates the window's visRgn. Unfortunately, it seems Apple goofed here. CalcVis only recalculates the visRgn of one window, while we need to recalculate the visRgns of all the windows following our window. Apple makes up for this, though, by supplying us with the routine CalcVisBehind (on same page as CalcVis). This function does not only recalculate the visRgn of our window, but also of all the windows following it. If you're curious, try replacing CalcVisBehind by CalcVis and watch what happens on the screen (don't do this if you dislike making a mess of things).

Another thing: the above call to PaintOne paints the whole window white, and not only the newly exposed part. Therefore, I've created the utility function BringForward, that calls SendBehind, PaintOne and CalcVisBehind, and only repaints those newly exposed parts. BringForward is called by TSelectWindow and many other routines in the TWindow Manager.

If you look at the source code for TSelectWindow, you'll see it does a lot more than just calling BringForward. This is because we have to take desk accessories into account. An open desk accessory will always lie above any tool windows when it is active. If the user selects one of the document windows, it naturally comes to the front, but not until all of the tool windows have also been brought to the front (remember that they were lying behind the desk accessory). I've written another utility function for this, obviously called BringToolsForward. This function makes repeated calls to BringForward for each visible tool window. These calls it makes to BringForward have a special flag to tell it not to call CalcVisBehind each time. This is because CalcVisBehind is a relatively time-consuming function. Once all the tool windows have been brought forward, BringToolsForward makes one final call to CalcVisBehind, recalculating all those changed visRgns at once. This speeds things up by about 30%, which is nice when you have a lot of tool windows (at one time during testing, I created about 50). The speed bottleneck, however, seems to be SendBehind. I wish Apple would publish interfaces to the Layer Manager in MultiFinder, because it has much more efficient ways of quickly moving windows to the front. Those aren't standard Window Manager functions working there ...

Two other useful utility functions internal to the TWindow Manager are NextVisWindow and PrevVisWindow. They operate fairly straightforward, and do exactly what their names imply: NextVisWindow returns the next visible window, and PrevVisWindow returns the previous visible window in the window list, with respect to the window that gets passed to it. NextVisWindow and PrevVisWindow are used in various places in the TWindow Manager, including here in TSelectWindow.

Dragging Windows

TDragWindow replaces the standard toolbox function DragWindow. It lets you drag a document window, while ensuring that it doesn't obscure any tool windows that may be lying in front of it. Again, I am grateful to Apple for doing the dirty work: they wrote this useful low level function called DragGrayRgn (Inside Macintosh page I-294) that pulls a gray outline of a region around. DragWindow actually calls DragGrayRgn, but in the wrong way (for our purposes).

The first thing TDragWindow does is check if the user is holding down the Command key. It gets this information from the EventRecord that you pass it. (Note that TDragWindow wants you to pass an EventRecord as its second parameter, while DragWindow wants the point where the mouse was clicked.) If the Command key is not being held down, TDragWindow calls TSelectWindow to move the window to the front (while staying below any tool windows, remember?). Once that is done, we can call DragGrayRgn to move an outline of the window. This outline is the window's strucRgn, so we pass copy of the strucRgn to DragGrayRgn. DragGrayRgn needs some more parameters, like the point the mouse was originally clicked and the bounding rectangles for the drag, but they are pretty obvious from the documentation in Inside Macintosh.

We have one last problem: how do we get DragGrayRgn to draw its outline below the tool windows? Aha, well, there's yet another low-level Window Manager function for that (I hope I'm not boring you). It's called ClipAbove, and Inside Macintosh says that "ClipAbove sets the clipRgn of the Window Manager

port to be the desktop intersected with the current clipRgn, minus the structure regions of all the windows in front of the given window." Great. So all we have to do is set the current port to the Window Manager port, set its clipRgn to the whole current desktop, and call ClipAbove. Then we go ahead and call DragGrayRgn. You'll see that the gray outline gets drawn in the Window Manager port, and stays neatly below the tool windows. After dragging, we check if the user really did move the window by examining DragGrayRgn's result. If he did, we call MoveWindow to actually move the window to its new position. And of course we restore the Window Manager port's clipRgn (since we've been messing with it), and reset the current port.

Dealing With Activate and Deactivate Events

Normally, the Window Manager in the Macintosh ROM posts an deactivate event when the frontmost window gets moved back, and a activate event when a window gets moved to the front. This won't work in our case, because our document windows never get moved to the front. They always stay behind our tool windows. But we still want selected document windows to receive (de)activate events, and preferably in the standard way: as events returned by some kind of TGetNextEvent function. My solution was to copy the mechanism Apple uses: the Macintosh has two low memory globals, CurActivate and CurDeactive. GetNextEvent checks these two globals first, before it examines the event queue. If it finds that one of these globals is nonzero, it fills in the user's EventRecord with an activate event for the window pointed to by one of these globals. CurActivate holds a pointer to the window that needs to be activated, and CurDeactive a pointer to the window that needs to be deactivated. Deactivate events have a higher priority than activate events.

My new TGetNextEvent does more or less the same, but instead it uses my globals toBeActivated and toBeDeactivated. (Please note that they are not in low memory, so don't worry about using them.) Whenever any routine in the TWindow Manager moves windows around, it updates these two globals. When TGetNextEvent gets called (as a replacement for GetNextEvent - writing TWaitNextEvent is left as an exercise for the reader), it checks these two globals and returns its own activate events back to the calling application.

TGetNextEvent handles three types of situations. It calls EventAvail to see what kind of event is currently pending. (1) The first situation arises when there is a true (de)activate event pending. In that case, it gets the event with GetNextEvent (the real one). Depending on the type of event (activate or deactivate), it compares the event's window with either the toBeActivated or the toBeDeactivated global. If the window the toolbox wants to activate is equal to toBeActivated, or the window it wants to deactivate is equal to toBeDeactivated, the toolbox is doing fine and there is nothing left for us to rectify. If toBeActivated or toBeDeactivated are nil, it means we didn't anticipate the activate event. This is an important occurrence and happens when a desk accessory suddenly gets opened or closed. In this case we substitute our front document window (frontDocWindow) for the window that gets (de)activated; additionally, we call HiliteWindow to highlight or unhighlight the frontDocWindow's title bar, since that hasn't been done yet, and we call HiliteTools to unhighlight or highlight all the tool windows. HiliteTools is a simple utility function I wrote that traverses all the tool windows, and calls HiliteWindow for each visible one. Oh yes, if there wasn't any document window open to pass the event to, we call TGetNextEvent again (recursively), thereby discard the current (de)activate event, and get a new event. After handling the event we reset toBeActivated or toBeDeactivated to nil and return.

(2) The second situation occurs when EventAvail does not return an (de)activate event and either toBeDeactivated or toBeActivated is not nil. This means some action of the TWindow Manager generated an (de)activate event without the toolbox knowing about it (like when you call TSelectWindow to activate a window without bringing it completely to the front). The thing to do is to fill the caller's EventRecord with an activate event for the window in the toBeActivated or toBeDeactivated global. The call to WindowExists has to do with dialog windows: I'll get to that later. (3) The third and final situation occurs when EventAvail doesn't return an (de)activate event, and both toBeActivated and toBeDeactivated are nil. There's nothing out of the ordinary to do here, so we simply call GetNextEvent and return the event.

While I was debugging this routine, I discovered something I had not anticipated since it had not been mentioned in Inside Macintosh: now and then, activate events were being returned for desk accessories! I was expecting GetNextEvent to take care of this by calling SystemEvent to pass the event on to the desk accessory directly (see page I-442). To remedy this I included the extra test in TGetNextEvent on the type of window belonging to an activate event. If the window isn't one of our (tool or document-) windows, we leave the event as it is. Another surprise were the deactivate events that were being posted for invisible windows. But if you think about it, it makes sense. When you hide the front window, the toolbox moves the window behind it to the front and activates it. Should you make your hidden window visible again, it will appear behind the current frontmost window, and should thus appear deactivated. Therefore the deactivate event. It makes you realize a lot is happening behind the scenes ...

Adding Windows

At first I thought it would be easy to write front ends for functions like NewWindow and GetNewWindow. All I would have to do was make sure that the calling application wasn't trying to create new document windows in front of existing tool windows. I thought I could check this with a couple of nested if-else statements. I managed to put together some stuff, but it didn't seem to cover everything. Whenever I thought about the code for a while, I would come up with a strange situation (usually in the bathroom, does that happen to you too?) that I hadn't anticipated yet. Things went really wrong when I tried to write TShowWindow! There were so many different combinations of windows, I had to take a completely different approach. The important thing to realize is that once the application program starts creating new windows, or making windows visible,

these windows can appear absolutely anywhere in the window list. A document window could appear on top, with tools behind it. Or a desk accessory might turn up between a tool and a document window, which is also not allowed. When calling GetNewWindow or NewWindow, windows can turn up in the wrong place, because the caller can specify any position for the new window. When calling ShowWindow, things can go wrong, because you have very little control over what is happening with your invisible windows. OK, I admit I make things worse myself, because the BringToolsForward function, for example, only moves visible tool windows, and leaves invisible tools scattered all over the window list. But it is more efficient to keep only the visible windows in their correct planes. We just have to be very careful when we make windows visible again.

First of all, I changed my earlier TGetNewWindow and TNewWindow into very simple functions that do two things: (1) They create the window as specified by their parameters, but make it invisible. They then set the window's windowKind field according to an extra TNewWindow or TGetNewWindow parameter. This parameter is also called windowKind, and lets the application indicate if it wants a tool window or document window. (Remember that the TWindow Managers distinguishes tool windows by looking at the windowKind field in the WindowRecord.) But the important thing is that the window is always created invisible. (2) If the caller wants the window to be visible, I call the function TShowWindow afterwards. TShowWindow does the tedious work of making the window show up in an allowed position, and all the testing code is centralized in this one function. So how do I find out if the caller wants a visible or invisible window? Well, when the calling application uses TNewWindow it's easy: I just examine the visible parameter. With TGetNewWindow I have do some extra work, because there is no visible parameter, but only a WIND resource. TGetNewWindow reads the WIND resource manually with a GetResource call, and gets the visible field from there. While the resource is in memory, TGetNewWindow also reads the other fields from it, such as the title, goAwayFlag etc. It then calls ReleaseResource to get rid of the WIND template, and creates the window with a normal NewWindow call.

Removing Windows

Before I get into TShowWindow, I want to point out that removing windows is very similar to adding windows. TCloseWindow and TDisposeWindow are very simple functions that call THideWindow to hide the window and make sure the rest of the windows are still in allowed positions. They then get rid of the window data structures through the usual calls to CloseWindow and DisposeWindow. There is one tricky situation though: suppose the window is the active one. THideWindow will then post a deactivate event for it in toBeDeactivated. If the window gets destroyed, that event will have to be removed. That's why I check if the window is equal to toBeDeactivated, and if it is, clear toBeDeactivated. The toolbox does the same thing (hardly surprising, because that's where I got the idea in the first place): CloseWindow and DisposeWindow check CurDeactive and clear it if necessary.

So it all boils down to THideWindow and TShowWindow - on to the next section, class.

Hiding and Showing - Where Are We In the First Place?

If you start thinking about it, many strange permutations of windows can occur on the desktop: desk accessories, tool windows, document windows, modal dialogs and modeless dialogs. Somehow, we have to keep the tools in front of the documents and modeless dialogs, while letting DAs and modal dialogs be on top from time to time. And any category of windows might be absent - suppose the application doesn't have any tool windows visible?

There may be many window permutations, but you will be pleased to hear the number is not infinite. In fact, there are precisely ten distinct positions a window can be in with respect to the other windows on the screen. (There may be more or less, depending on your own way of classifying windows.) Once you know in what position a particular window is, it's very clear what you have to do when that window is hidden or shown. The utility function TGetWPosition finds out what this position is: THideWindow or TShowWindow pass it a window, and it will tell them in which of the ten places the window is. It returns one of the ten possible constants describing this position. Note that TGetWPosition works for both visible and invisible windows. Since these ten positions are central to understanding how THideWindow and TShowWindow work, let me list them for you (remember that our window can be visible or invisible, and can be either a tool-, document- or dialog window). You'll want to glance at figure 2 along the way.

alone there are no other visible windows on the screen

beforeDialog our window is completely in front, and the window behind it is a modal dialog

beforeSystem our window is completely in front, and the window behind it is a desk accessory

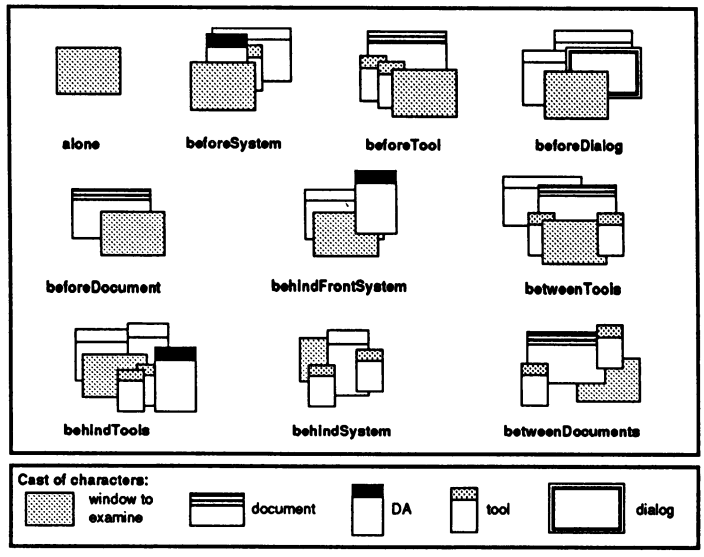


Figure 2: Window Positions

beforeTool our window is completely in front, and the window behind it is a tool window

beforeDocument our window is completely in front, and the window behind it is a document window

behindFrontSystem there are one or more desk accessories in front, and our window is lying just behind them (but before any tool or document windows); it could be our window is the frontmost tool window

betweenTools our window is lying among the tool windows; if it's visible, it could also be the last tool window

behindTools our window is lying behind all the tool windows, but in front of any document windows; if our window is a tool, it can only be invisible, otherwise it would be in the betweenTools position

behindSystem this is really an obscure arrangement, and only exists when our window is invisible: there are no document windows, there is a tool in front, and the tools are followed by one or more desk accessories; our invisible window is lying behind the desk accessories

betweenDocuments our window is lying among the document windows; if it's a visible document window, it certainly isn't the frontmost one

TGetWPosition starts with a pointer to the frontmost visible window, and works its way through the window list until it finds our window. While it's doing this, it keeps track of current window position. Once it hits our window, it returns this position. Notice that when our window is visible, TGetWPosition really looks for the window preceding it. That way the algorithm works for both visible and invisible windows. As a bonus you also get the windowKind of the frontmost window in one of the parameters. THideWindow and TShowWindow might want to look at this to see if there is a desk accessory in front.

We've Found Our Place!

Now everything falls into place. TShowWindow and THideWindow simply call TGetWPosition, and let three big switch statements (the first if we are dealing with a tool window, the second for a document window/modeless dialog and a little one for a modal dialog) handle the nitty gritty details. We have a case for almost every window position. Some cases are the same (like hiding a tool in the positions alone and beforeSystem), while others are so obscure that they cannot possibly occur in each function (there is no case for hiding a window in the behindSystem position). If you look at the source, you'll see that each case executes a small but different block of code. The code deals with things like setting the toBeActivated and toBeDeactivated globals, bringing tools forward with BringToolsForward, updating frontToolWindow, backToolWindow and frontDocWindow, and sometimes highlighting windows manually with HiliteWindow and HiliteTools. Most cases are easy to follow, although there are a few complex ones. The behindTools case for a hiding a document window does a lot of things, so it is a nice example to illustrate how all of this works.

An Example of Hiding a Window

When our document window is behindTools, it is obviously the frontmost document window (it is lying just behind the tools). This means we have to find the new document window (our window is going to be hidden). We call NextVisWindow with a windowKind parameter of userKind. This means NextVisWindow will return the next visible document window (or modeless dialog, because TGetWKind considers modeless dialogs the same as document windows). But suppose the next visible window is a desk accessory? That's why there is the other call to NextVisWindow. This call determines what the real first window behind ours is, regardless of the type of the window, because we use the anyKind parameter. We then hide our window with ShowHide.

If there is no window behind ours, we only have to post a deactivate event for our hidden window. If there is, there are two possibilities: it is a desk accessory - if behindWindow differs from frontDocWindow - or it is a document window, in which case behindWindow should be equal to frontDocWindow. A desk accessory will have to be brought completely to the front - if there isn't already a DA in front - with SelectWindow. Then something important: a document window that could be lying much further back (behind the DA) has to be brought forward, just behind the tool windows. Tools always "stick" to at least one document window. This is what the BringForward call is for. Finally, if the window behind ours was not a DA, but a document window, we make it the active window by highlighting it and posting the necessary activate events. All of this only if there isn't a DA in front, of course. Phew! Rest assured that if you understood this, the rest of the cases are a piece of cake.

TNewDialog, TGetNewDialog, TCloseDialog in TDisposDialog?

Don't panic! I'm not going to write front ends for every manager in the toolbox. You will have to do a few extra things for dialogs though (especially to get the window behind it to deactivate properly). When you want to put up a new dialog, do the following: first call NewDialog or GetNewDialog to create the dialog, but make sure it is still invisible. Then call TShowWindow to make it visible. TShowWindow will ensure that the document window behind it gets deactivated properly, and that all the tools are unhighlighted. Chances are that you are already doing this (creating an invisible dialog, and showing it afterwards) to install a user item to boldly go where no man ... oops, to boldly outline the OK button in the dialog. To remove a dialog you should do something similar: call THideWindow to hide it, and CloseDialog or DisposDialog afterwards.

You might remember that when a window gets destroyed, a possible pending deactivate event for it should be removed as well. Unfortunately, THideWindow posts a deactivate event in toBeDeactivated, but it never gets removed. Sure, CloseDialog and DisposDialog remove the toolbox's event in CurDeactive, but what about our own event? I suppose I could have written front ends for these Dialog Manager functions, but I think I've gone far enough as is. So this is where the WindowExists function comes in (remember, I mentioned it back when I was discussing TGetNextEvent). As a last resort, WindowExists gets

called to make sure the window that needs deactivating still exists. If it doesn't, we call TGetNextEvent again to get a new event instead.

The Point Of It All

The advantage of all this tedious, boring, dense and voluminous code is that you can forget it. Yes, go ahead and forget it. Once the TWindows.c module has been compiled, you can link it with your application and you won't have to worry about tool windows again. Just change your usual calls to the Window Manager into calls to the TWindow Manager. (See TWindowTester for examples). Whenever you create new document or tool windows, hide them, drag them around the screen, or bring them to the front, the TWindow Manager will do its utmost best to keep you happy by keeping those tools floating above your documents.

There is far more information in the source files than I have mentioned here. Especially the sources on the source code disks have lots more comments and explanations than what would fit in the magazine. Still, I might have overlooked something. In that case (or for any not necessarily useful reason - what I mean to say is: I don't mind getting mail), you can always reach me at one of the addresses below:

Thomas Fruin	fruin@hlerul5.BITNET
Galgewater 38	thomas@uvabick.UUCP
2311 VZ LEIDEN	dibs@well.UUCP
The Netherlands	hol0066 on AppleLink
	2:508/15 on FidoNet

Listing: TWindow.c

```
/*
 *  TWindows.c
 *
 *  C source of an extended Window Manager that
 *  supports tool  windows; these are windows
 *  that always float on top, for palettes and
 *  tools.
 *
 *  Written in MPW C 2.0
 *
 *  Copyright Thomas Fruin 1988
 *  All rights reserved.
 */

/*
 *  T O O L B O X   I N C L U D E S
 */
#include <Types.h>
#include <Memory.h>
#include <Quickdraw.h>
#include <Events.h>
#include <Windows.h>
#include <ToolUtils.h>
#include <OSUtils.h>
#include <Resources.h>

/*
 *  T O O L B O X   D E F I N I T I O N S
 */
#define GrayRgn     (*(RgnHandle *)0x9EE)
#define WindowList  (*(WindowPeek *)0x9D6)
/*
 *  D E F I N I T I O N S
 */
  /* Defines LONGINT, INTEGER, the windowKind
     constants and the externally callable
     TWindow Manager functions. */

#include <TWindows.h>
  /* Result codes describing window positions, as
     returned by the function GetWPosition. */
#define alone             0
#define beforeDialog      1
#define beforeSystem      2
#define beforeTool        3
#define beforeDocument    4
#define behindFrontSystem 5
#define betweenTools      6
#define behindTools       7
#define behindSystem      8
#define betweenDocuments  9

  /* Miscellaneous */

#define deactivFlag       0
#define noDrag       -32768
#define invisible     false
#define curVersion        1
#define postponeCalc   true
#define normalCalc    false
/*
 *  T Y P E S
 */
  /* The WINDData structure matches the structure
     of a WIND resource in a resource file.  It
     is used to access the various fields of the
     WIND resource after it has been read in from
     the file. */

struct WINDRecord
  {
    Rect      boundsRect;
    INTEGER   procID;
    Boolean   visible;
    char      filler1;
    Boolean   goAwayFlag;
    char      filler2;
    LONGINT   refCon;
    Str255    title;
  };
typedef struct WINDRecord WINDData;
typedef struct WINDRecord *WINDPtr;
typedef struct WINDRecord **WINDHandle;

  /* The WDEFHeader structure matches the
     structure of the (optional) header of a
     window definition procedure resource. */

struct WDEFRecord
  {
    INTEGER   branch;
    INTEGER   flags;
    LONGINT   type;
    INTEGER   ID;
    INTEGER   version;
  };
typedef struct WDEFRecord WDEFHeader;
typedef struct WDEFRecord *WDEFPtr;
typedef struct WDEFRecord **WDEFHandle;

/*
 *  G L O B A L S
 */
static WindowPeek toBeActivated,
                  toBeDeactivated,
                  frontToolWindow,
                  backToolWindow,
                  frontDocWindow;
static RgnHandle  firstRgn,
```

```
                secondRgn;
static SysEnvRec  theWorld;
/*
 *  T I N I T W I N D O W S
 */
void
TInitWindows()
{
  OSErr   err;

  frontToolWindow =
  backToolWindow  =
  frontDocWindow  =
  toBeDeactivated =
  toBeActivated   = nil;
  firstRgn  = NewRgn();
  secondRgn = NewRgn();
  err = SysEnvirons( curVersion, &theWorld );
}
/*
 *  T G E T N E W W I N D O W
 */

WindowPtr
TGetNewWindow( windowKind, windowID,
               wStorage, behind )
  INTEGER     windowKind;
  INTEGER     windowID;
  Ptr         wStorage;
  WindowPtr   behind;
{
  Rect        boundsRect;
  Str255      title;
  Boolean     visible;
  INTEGER     procID;
  Boolean     goAwayFlag;
  LONGINT     refCon;
  WindowPtr   theWindow;
  WINDHandle  theWINDHandle;
  WINDPtr     theWINDPtr;

  theWINDHandle =
    ( WINDHandle )GetResource( 'WIND',windowID );
  if ( theWINDHandle != nil )
  {
    theWINDPtr = *theWINDHandle;
    boundsRect = theWINDPtr->boundsRect;
    visible    = theWINDPtr->visible;
    procID     = theWINDPtr->procID;
    goAwayFlag = theWINDPtr->goAwayFlag;
    refCon     = theWINDPtr->refCon;
    if ( theWINDPtr->title.length == 0 )
      title.length = 0;
    else
      BlockMove( &( theWINDPtr->title ), &title,
          ( Size )theWINDPtr->title.length + 1 );
    p2cstr( &title );

    ReleaseResource(( Handle )theWINDHandle );
    theWindow =
      NewWindow( wStorage, &boundsRect, &title,
                 invisible, procID, behind,
                 goAwayFlag, refCon );
    if ( theWindow != nil )
    {
      (( WindowPeek )theWindow )->windowKind =
          windowKind;
      if ( visible != false )
        TShowWindow( theWindow );
    }
  }
  else
    theWindow = nil;
  return( theWindow );
}
/*
 *  T N E W W I N D O W
 */

WindowPtr
TNewWindow( windowKind, wStorage, boundsRect,
            title, visible, procID, behind,
            goAwayFlag, refCon )
  INTEGER     windowKind;
  Ptr         wStorage;
  Rect        *boundsRect;
  char        *title;
  Boolean     visible;
  INTEGER     procID;
  WindowPtr   behind;
  Boolean     goAwayFlag;
  LONGINT     refCon;
{
  WindowPtr theWindow;

  theWindow =
    NewWindow( wStorage, boundsRect, title,
               invisible, procID, behind,
               goAwayFlag, refCon );
  if ( theWindow != nil )
  {
    (( WindowPeek )theWindow )->windowKind =
          windowKind;
    if ( visible != false )
      TShowWindow( theWindow );
  }
  return( theWindow );
}
/*
 *  T C L O S E W I N D O W
 */

void
TCloseWindow( theWindow )
  WindowPtr   theWindow;
{
  THideWindow( theWindow );
  CloseWindow( theWindow );

  if ( theWindow == toBeDeactivated )
    toBeDeactivated = nil;
}
/*
 *  T D I S P O S E W I N D O W
 */

void
TDisposeWindow( theWindow )
  WindowPtr   theWindow;
{
  THideWindow( theWindow );
  DisposeWindow( theWindow );

  if ( theWindow == toBeDeactivated )
    toBeDeactivated = nil;
}


/*
 *  T S E L E C T W I N D O W
 */

void
TSelectWindow( theWindow )
  WindowPtr   theWindow;
{
  void        BringForward();
  WindowPeek  thePWindow,
```

```
            theFrontWindow,
            NextVisWindow(),
            PrevVisWindow();
INTEGER     theWKind,
            frontWKind;
Boolean     toolsVisible,
            BringToolsForward();

    /* Initialize some variables. */

thePWindow     = ( WindowPeek )theWindow;
theFrontWindow = ( WindowPeek )FrontWindow();
theWKind       = TGetWKind( theWindow );
frontWKind     = TGetWKind(( WindowPtr )theFrontWindow );

if ( theWKind == toolKind )
{
  if ( frontWKind == systemKind )
  {
      /* Window is a tool window, with a desk
         accessory on top. */

    SelectWindow( theWindow );
    (void)BringToolsForward( thePWindow );
    if ( frontDocWindow != nil )
    {
      frontDocWindow->hilited = true;
      BringForward( frontDocWindow,
                    backToolWindow,
                    normalCalc );
      toBeActivated = frontDocWindow;
    }
    frontToolWindow = thePWindow;
  }
  else
  {
      /* Window is a tool window, and no desk
         accessory on top. */

    if ( thePWindow != frontToolWindow )
    {
      if ( thePWindow == backToolWindow )
      {
        backToolWindow =
          NextVisWindow( toolKind, thePWindow );
        if ( backToolWindow == nil )
          backToolWindow = PrevVisWindow( thePWindow );
      }
      BringToFront( theWindow );
      frontToolWindow = thePWindow;
    }
  }
}
else if ( theWKind == userKind )
{
  if ( frontWKind == systemKind )
  {
      /* Window is a document window (or a
         modeless dialog window), with a desk
         accessory on top. */

    toolsVisible =
      BringToolsForward(( WindowPeek )inFront );
    if ( toolsVisible == false )
      SelectWindow( theWindow );
    else
    {
      thePWindow->hilited = true;
      BringForward( theWindow,
                    backToolWindow,
                    normalCalc );
    }
    toBeActivated  = frontDocWindow = thePWindow;
  }
  else
  {
      /* Window is a document window (or a
         modeless dialog window), and no desk
         accessory on top. */

    if ( thePWindow != frontDocWindow )
    {
      if ( frontToolWindow == nil )
        SelectWindow( theWindow );
      else
      {
        HiliteWindow(( WindowPtr)frontDocWindow,
                                        false );
        thePWindow->hilited = true;
        BringForward( thePWindow,
                      backToolWindow,
                      normalCalc );
      }
      toBeDeactivated = frontDocWindow;
      toBeActivated   = frontDocWindow  = thePWindow;
    }
  }
}
/*
 *  T H I D E W I N D O W
 */

void
THideWindow( theWindow )
  WindowPtr   theWindow;
{
  WindowPeek  thePWindow,
              behindWindow;
  INTEGER     theWKind,
              thePosition,
              frontWKind,
              GetWPosition();
  void        HiliteTools();

  if ((( WindowPeek )theWindow )->visible
                                != false )
  {
    thePWindow  = ( WindowPeek )theWindow;
    theWKind    = TGetWKind( theWindow );
    thePosition = GetWPosition( thePWindow,
                                &frontWKind );

      /* Hide the window, and update the desktop
         according to the window's position. */

    if ( theWKind == toolKind )
    {
        /* The window is a tool window. */

      switch( thePosition )
      {
        case alone:
        case beforeSystem:
            HideWindow( theWindow );
            frontToolWindow =
            backToolWindow  = nil;
            break;

        case beforeTool:
            ShowHide( theWindow, false );
            frontToolWindow = ( WindowPeek )FrontWindow();
            break;

        case behindFrontSystem:
            ShowHide( theWindow, false );
            frontToolWindow =
              NextVisWindow( toolKind,
```


© The Definitive MacTutor, Vol. 4

```
                        thePWindow );
        if ( frontToolWindow == nil )
          backToolWindow = nil;
        break;

    case beforeDocument:
        frontToolWindow =
        backToolWindow  = nil;
        ShowHide( theWindow, false );
        break;

    /* NOTE: no behindTools possible! */

    case betweenTools:
        ShowHide( theWindow, false );
        if ( thePWindow == backToolWindow )
          backToolWindow = PrevVisWindow( thePWindow );
        break;
  }
}
else if ( theWKind == userKind )
{
    /* The window is a document window
       (or a modeless dialog window). */

  thePWindow->hilited = false;

  switch( thePosition )
  {
    case alone:
    case beforeSystem:
        toBeDeactivated = thePWindow;
        frontDocWindow  = nil;

    case betweenDocuments:
        HideWindow( theWindow );
        break;

    case beforeDocument:
        HideWindow( theWindow );
        toBeDeactivated = thePWindow;
        toBeActivated   =
        frontDocWindow  =
          ( WindowPeek )FrontWindow();
        break;

    case behindFrontSystem:
        ShowHide( theWindow, false );
        frontDocWindow =
          NextVisWindow( userKind,theWindow );
        break;

    case behindTools:
        frontDocWindow =
          NextVisWindow( userKind,theWindow);
        behindWindow =
          NextVisWindow( anyKind, theWindow );
        ShowHide( theWindow, false );
        if ( behindWindow == nil )
          toBeDeactivated = thePWindow;
        else
        {
          if ( behindWindow != frontDocWindow)
          {
            if ( frontWKind == toolKind )
            {
              HiliteTools( false );
              SelectWindow(( WindowPtr )
                           behindWindow );
              toBeDeactivated = thePWindow;
            }
            if ( frontDocWindow != nil )
              BringForward( frontDocWindow,
                            backToolWindow,
                            normalCalc );
          }
          else
          {
            if ( frontWKind == toolKind )
            {
              HiliteWindow( frontDocWindow,
                            true );
              toBeActivated = frontDocWindow;
              toBeDeactivated = thePWindow;
            }
          }
        }
        break;
    }
  }
  else
  {
      /* The window is a modal dialog window.*/

    switch( thePosition )
    {
      case alone:
      case beforeSystem:
      case beforeDialog:
          HideWindow( theWindow );
          break;

      case beforeTool:
      case beforeDocument:
          ShowHide( theWindow, false );
          HiliteTools( true );
          if ( frontDocWindow != nil )
          {
            toBeDeactivated = thePWindow;
            toBeActivated   = frontDocWindow;
            HiliteWindow(( WindowPtr)
                        frontDocWindow, true );
          }
          break;
    }
  }
}
/*
 *  T S H O W W I N D O W
 */

void
TShowWindow( theWindow )
  WindowPtr   theWindow;
{
  WindowPeek  thePWindow,
              theFrontWindow,
              behind,
              PrevVisWindow();
  INTEGER     theWKind,
              frontWKind,
              thePosition,
              GetWPosition();
  Boolean     toolsExist,
              BringToolsForward();
  void        BringForward(),
              HiliteTools();

  if ((( WindowPeek )theWindow )->visible
                                == false )
  {
          /* The window is still invisible. */

    thePWindow     = ( WindowPeek )theWindow;
    theFrontWindow = ( WindowPeek )FrontWindow();
    theWKind       = TGetWKind( theWindow );
    thePosition    = GetWPosition( thePWindow, &frontWKind );
```

```
if ( theWKind == toolKind )
{
    /* The window is a tool window.  Assume
       it needs to appear highlighted
       (correct this later). */

  thePWindow->hilited = true;

  switch( thePosition )
  {
    case beforeSystem:
        ShowWindow( theWindow );
        ( void )BringToolsForward(thePWindow);
        if ( frontDocWindow != nil )
        {
          frontDocWindow->hilited = true;
          BringForward( frontDocWindow,
                        backToolWindow,
                        normalCalc );
          toBeActivated = frontDocWindow;
        }
        frontToolWindow = thePWindow;
        break;

    case behindFrontSystem:
        thePWindow->hilited = false;

    case alone:
    case beforeDocument:
        if ( backToolWindow == nil )
           backToolWindow =  thePWindow;

    case beforeTool:
        ShowHide( theWindow, true );
        frontToolWindow = thePWindow;
        break;

    case behindTools:
        backToolWindow = thePWindow;

    case betweenTools:
        if ( frontWKind == systemKind )
          thePWindow->hilited = false;
        ShowHide( theWindow, true );
        break;

    case behindSystem:
        SendBehind( theWindow,
          ( WindowPtr )backToolWindow );
        if ( frontWKind == systemKind )
          thePWindow->hilited = true;
        ShowHide( theWindow, true );
        backToolWindow  = thePWindow;
        break;

    case betweenDocuments:
        if ( backToolWindow != nil )
          behind = backToolWindow;
        else
        {
          behind =
            PrevVisWindow( frontDocWindow );
          frontToolWindow = thePWindow;
        }
        if ( behind == nil )
          BringToFront( theWindow );
        else
        {
          SendBehind( theWindow, behind );
          if ( frontWKind == systemKind )
            thePWindow->hilited = false;
        }
        ShowHide( theWindow, true );
        backToolWindow = thePWindow;
        break;
  }
}
else if ( theWKind == userKind )
{
    /* The window is a document window. */

  switch( thePosition )
  {
    case beforeSystem:
        toolsExist = BringToolsForward(
               ( WindowPeek )inFront );
        if ( toolsExist != false )
          thePWindow->hilited = true;

    case alone:
        ShowWindow( theWindow );
        toBeActivated  = frontDocWindow = thePWindow;
        break;

    case beforeTool:
        if ( frontDocWindow != nil )
        {
          HiliteWindow(( WindowPtr )
                  frontDocWindow, false );
          toBeDeactivated = frontDocWindow;
        }
        SendBehind( theWindow,
                    backToolWindow );
        thePWindow->hilited = true;
        ShowHide( theWindow, true );
        toBeActivated  = frontDocWindow = thePWindow;
        break;

    case beforeDocument:

          /* The three steps HiliteWindow,
             HiliteWindow and ShowHide could
             be replaced with one call to
             ShowWindow, but this gives a
             better visual effect. (Try the
             difference.) */

        HiliteWindow( frontDocWindow, false );
        HiliteWindow( theWindow, true );
        ShowHide( theWindow, true );
        toBeDeactivated = frontDocWindow;
        toBeActivated   = frontDocWindow  = thePWindow;
        break;

    case behindFrontSystem:
        if ( backToolWindow != nil )
          SendBehind( theWindow,
              ( WindowPtr )backToolWindow );
        ShowHide( theWindow, true );
        frontDocWindow  = thePWindow;
        break;

    case betweenTools:
        SendBehind( theWindow,
              ( WindowPtr )backToolWindow );

    case behindTools:
        if ( frontWKind == toolKind )
        {
          thePWindow->hilited = true;
          toBeActivated = thePWindow;
          if ( frontDocWindow != nil )
          {
            HiliteWindow(( WindowPtr )
                frontDocWindow, false );
            toBeDeactivated = frontDocWindow;
          }
```


© The Definitive MacTutor, Vol. 4

```
            }
            ShowHide( theWindow, true );
            frontDocWindow  = thePWindow;
            break;

        case behindSystem:
            SendBehind( theWindow,
                  ( WindowPtr )backToolWindow );
            if ( frontWKind == toolKind )
            {
              thePWindow->hilited = true;
              toBeActivated = thePWindow;
            }
            ShowHide( theWindow, true );
            frontDocWindow  = thePWindow;
            break;

        case betweenDocuments:
            ShowHide( theWindow, true );
            break;
      }
    }
    else
    {
        /* The window is a modal dialog window.*/

      switch( thePosition )
      {
        case alone:
        case beforeSystem:
        case beforeDialog:
            ShowWindow( theWindow );
            break;

        case beforeTool:
        case beforeDocument:
            HiliteTools( false );
            if ( frontDocWindow != nil )
            {
              toBeDeactivated = frontDocWindow;
              HiliteWindow(( WindowPtr )
                    frontDocWindow, false );
            }
            ShowHide( theWindow, true );
            break;
      }
    }
  }
}
/*
 *  T F R O N T W I N D O W
 */

WindowPtr
TFrontWindow( wantedKind )
  INTEGER     wantedKind;
{
  WindowPtr   theWindow;
  INTEGER     theWKind;

  theWindow = FrontWindow();
  theWKind  = TGetWKind( theWindow );

  if ( theWKind == systemKind  ||
       theWKind == dialogKind )
    return( theWindow );
  else
    switch ( wantedKind )
    {
      case toolKind:
          return(( WindowPtr )frontToolWindow );
          break;

      case userKind:
          return(( WindowPtr )frontDocWindow );
          break;

      case anyKind:
          return( theWindow );
          break;
    }
}
/*
 *  T D R A G W I N D O W
 */

void
TDragWindow( theWindow, theEvent, boundsRect )
  WindowPtr   theWindow;
  EventRecord *theEvent;
  Rect        *boundsRect;
{
  GrafPtr     savePort,
              wPort;
  Rect        limitRect,
              slopRect;
  RgnHandle   desktopRgn;
  LONGINT     result;
  INTEGER     hDrag,
              vDrag;
  Point       thePoint;

  if (( theEvent->modifiers & cmdKey ) == false )
  {
    TSelectWindow( theWindow );
    if ( StillDown() == false )
      return;
  }

  GetPort    ( &savePort );
  GetWMgrPort( &wPort );
  SetPort    (  wPort );

  if ( theWorld.machineType == envMac  ||
       theWorld.machineType == envXL  )
    desktopRgn = GrayRgn;
  else
    desktopRgn = GetGrayRgn();

  GetClip  ( firstRgn );
  SetClip  ( desktopRgn );
  ClipAbove(( WindowPeek )theWindow );

  CopyRgn(( WindowPeek )theWindow )->strucRgn,
                                    secondRgn );
  SetPt  ( &thePoint, theEvent->where.h,
                      theEvent->where.v );
  GlobalToLocal( &thePoint );
  result = DragGrayRgn( secondRgn, &thePoint,
                boundsRect, boundsRect,
                noConstraint, ( ProcPtr )nil );

  SetClip( firstRgn );

  vDrag = HiWord( result );
  hDrag = LoWord( result );
  if ( !( vDrag == noDrag && hDrag == noDrag ))
  {
    SetPort( theWindow );
    SetPt  ( &thePoint,
             theWindow->portRect.left,
             theWindow->portRect.top );
    LocalToGlobal( &thePoint );
    MoveWindow( theWindow, thePoint.h + hDrag,
                  thePoint.v + vDrag, false );
  }

  SetPort( savePort );
```

```
  SetEmptyRgn( firstRgn );
  SetEmptyRgn( secondRgn );
}
/*
 *  T G E T N E X T E V E N T
 */

Boolean
TGetNextEvent( eventMask, theEvent )
  INTEGER   eventMask;
  EventRecord *theEvent;
{
  Boolean   result,
            WindowExists();
  INTEGER   theWKind;
  void      HiliteTools();

  if ( EventAvail( eventMask, theEvent )
             != false  &&
       theEvent->what == activateEvt )
  {
    result = GetNextEvent( eventMask, theEvent );
    theWKind = TGetWKind(( WindowPtr )theEvent->message );

    if (( theEvent->modifiers & activeFlag )
          == false )
    {
      if ( theEvent->message != ( LONGINT )toBeDeactivated )
      {
        if ( toBeDeactivated != nil )
          theEvent->message = ( LONGINT )toBeDeactivated;
        else
        {
          if ( theWKind == userKind  ||
               theWKind == toolKind  )
          {
            if ( theWKind == toolKind )
              HiliteTools( false );
            if ( frontDocWindow != nil )
            {
              HiliteWindow((WindowPtr)frontDocWindow, false );
              theEvent->message  = ( LONGINT )frontDocWindow;
            }
            else
              result = TGetNextEvent( eventMask,theEvent );
          }
        }
      }
      toBeDeactivated = nil;
    }
    else
    {
      if ( theEvent->message
            != ( LONGINT )toBeActivated )
      {
        if ( toBeActivated != nil )
          theEvent->message =
            ( LONGINT )toBeActivated;
        else
        {
          if ( theWKind == userKind  ||
               theWKind == toolKind  )
          {
            if ( theWKind == toolKind )
              HiliteTools( true );
            if ( frontDocWindow != nil )
            {
              HiliteWindow(( WindowPtr )
                  frontDocWindow, true );
              theEvent->message  =
                ( LONGINT )frontDocWindow;
            }
            else
              result = TGetNextEvent( eventMask,theEvent );
          }
        }
      }
      toBeActivated = nil;
    }
  }
  else
  {
    if ( toBeDeactivated != nil         &&
         ( eventMask & activMask ) != false )
    {
      if ( WindowExists( toBeDeactivated )
             != false )
      {
        theEvent->what      = activateEvt;
        theEvent->message   = ( LONGINT )toBeDeactivated;
        theEvent->modifiers = deactivFlag;
        toBeDeactivated     = nil;
        result = true;
      }
      else
      {
        toBeDeactivated   = nil;
        result = TGetNextEvent( eventMask, theEvent );
      }
    }
    else if ( toBeActivated != nil        &&
              ( eventMask & activMask) != false )
    {
      theEvent->what      = activateEvt;
      theEvent->message   = ( LONGINT )toBeActivated;
      theEvent->modifiers = activeFlag;
      toBeActivated       = nil;
      result = true;
    }
    else
    {
      result = GetNextEvent( eventMask,theEvent);
    }
  }

  return( result );
}
/*
 *  W I N D O W E X I S T S
 */

Boolean
WindowExists( thePWindow )
  WindowPeek  thePWindow;
{
  WindowPeek  loopWindow;
  Boolean     done;

  for ( done = false,
          loopWindow = WindowList;
      done == false  &&  loopWindow != nil;
      loopWindow = loopWindow->nextWindow )
  {
    if ( loopWindow == thePWindow )
      done = true;
  }
  return( done );
}

/*
 *  G E T W P O S I T I O N
 */

static INTEGER
GetWPosition( thePWindow, frontWKind )
  WindowPeek  thePWindow;
  INTEGER     *frontWKind;
{
```

```
  WindowPeek  theFrontWindow,
              newFrontWindow,
              loopWindow,
              NextVisWindow();
  Boolean     done;
  INTEGER     result,
              loopWKind;

  theFrontWindow = ( WindowPeek )FrontWindow();

  if ( thePWindow->visible != false &&
       theFrontWindow == thePWindow )
    newFrontWindow = NextVisWindow( anyKind, theFrontWindow );
  else
    newFrontWindow = theFrontWindow;

  if ( newFrontWindow == nil )
    return( alone );
  *frontWKind = TGetWKind(( WindowPtr )newFrontWindow );

  if ( thePWindow->visible != false &&
       theFrontWindow == thePWindow ||
       theFrontWindow == NextVisWindow( anyKind, thePWindow ))
  {
      /* Our window is visible and the frontmost,
         or it's invisible and lying in front of
         the frontmost visible window. */

    if ( *frontWKind == systemKind )
      result = beforeSystem;
    else if ( *frontWKind == dialogKind )
      result = beforeDialog;
    else if ( *frontWKind == toolKind )
      result = beforeTool;
    else
      result = beforeDocument;
  }
  else
  {
    if (thePWindow->visible != false )
      thePWindow = PrevVisWindow( thePWindow );

    for ( done = false,
            loopWindow = theFrontWindow;
          done == false;
          loopWindow = loopWindow->nextWindow )
    {
      loopWKind =
        TGetWKind(( WindowPtr )loopWindow );
      if ( loopWindow == theFrontWindow  &&
           loopWKind  == systemKind     )
        result = behindFrontSystem;
      else if ( loopWindow == frontToolWindow &&
           backToolWindow != frontToolWindow  )
        result = betweenTools;
      else if ( loopWindow == backToolWindow )
        result = behindTools;
      else if ( loopWKind == systemKind       &&
                loopWindow->visible != false  &&
                result == behindTools )
        result = behindSystem;
      else if ( loopWindow == frontDocWindow )
        result = betweenDocuments;

      if ( loopWindow == thePWindow )
        done = true;
    }

    return( result );
  }
}
/*
 *  T G E T W K I N D
 */

INTEGER
TGetWKind( theWindow )
  WindowPtr   theWindow;
{
  INTEGER     theWKind,
              varCode,
              windowID;
  WDEFHandle  defProc;

  if ( theWindow != nil )
  {
    theWKind =
      (( WindowPeek )theWindow )->windowKind;

    if ( theWKind <= systemKind )
      theWKind = systemKind;
    else if ( theWKind == dialogKind )
    {
      defProc =
        ( WDEFHandle )
        ((WindowPeek )theWindow )->windowDefProc;

      if ( theWorld.machineType == envMac  ||
           theWorld.machineType == envXL  )
        varCode  = ( LONGINT )defProc >> 24 && 0x0000000F;
      else
        varCode = GetWVariant( theWindow );

      if (   defProc != nil   &&
           *defProc != nil   &&
         ( *defProc )->type == 'WDEF' )
        windowID =
          ( *defProc )->ID * 16 + varCode;
      else
        windowID = dBoxProc;

      if ( windowID == documentProc   ||
           windowID == noGrowDocProc  ||
           windowID == zoomDocProc     )
        theWKind = userKind;
    }
    else if ( theWKind == toolKind )
      /* nothing */ ;
    else if ( theWKind >= userKind )
      theWKind = userKind;
  }
  return( theWKind );
}
/*
 *  B R I N G T O O L S F O R W A R D
 */

static Boolean
BringToolsForward( thePWindow )
  WindowPeek  thePWindow;
{
  WindowPeek  NextVisWindow(),
              toolWindow,
              loopWindow;
  void        BringForward();

  if ( thePWindow == ( WindowPeek )inFront )
  {
    toolWindow = NextVisWindow( toolKind,
                 ( WindowPeek )FrontWindow() );

    if ( toolWindow != nil )
    {
      loopWindow      = toolWindow->nextWindow;
      SelectWindow(( WindowPtr )toolWindow );
      frontToolWindow = backToolWindow  = toolWindow;
    }
    else
```

```
    (
      return( false );
    }
  }
  else
  {
    loopWindow     = thePWindow->nextWindow;
    backToolWindow = thePWindow;
  }

  SetEmptyRgn( secondRgn );
  thePWindow = nil;

  while ( loopWindow != nil )
  {
    if ( loopWindow->windowKind == toolKind  &&
         loopWindow->visible    != false   )
    {
      toolWindow = loopWindow;
      toolWindow->hilited = true;
      BringForward( toolWindow, backToolWindow,
                                postponeCalc );
      UnionRgn( secondRgn, toolWindow->strucRgn,
                secondRgn );
      backToolWindow = toolWindow;
      if ( thePWindow == nil )
        thePWindow = toolWindow;
    }
    loopWindow = loopWindow->nextWindow;
  }

  CalcVisBehind( thePWindow, secondRgn );
  SetEmptyRgn( secondRgn );

  return( true );
}

/* End of BringToolsForward */
/*
 *  N E X T V I S W I N D O W
 */

static WindowPeek
NextVisWindow( wantedKind, thePWindow )
  INTEGER     wantedKind;
  WindowPeek  thePWindow;
{
  WindowPeek  loopWindow,
              wantedWindow;

  if ( thePWindow == ( WindowPeek )inFront )
  {
    thePWindow = ( WindowPeek )FrontWindow();
    if ( thePWindow == nil )
      return( thePWindow );
    else if ( wantedKind == anyKind   ||
              wantedKind ==TGetWKind(( WindowPtr)thePWindow ))
      return( thePWindow );
  }

  for ( wantedWindow = nil,
          loopWindow = thePWindow->nextWindow;
        wantedWindow == nil  &&
          loopWindow != nil;
        loopWindow = loopWindow->nextWindow )
  {
    if ( loopWindow->visible != false )
      if ( wantedKind == anyKind    ||
           wantedKind ==
             TGetWKind(( WindowPtr )loopWindow ))
        wantedWindow = loopWindow;
  }

  return( wantedWindow );
}
/*
 *  P R E V V I S W I N D O W
 */

static WindowPeek
PrevVisWindow( thePWindow )
  WindowPeek  thePWindow;
{
  WindowPeek  loopWindow,
              prevWindow,
              theFrontWindow,
              NextVisWindow();
  Boolean     done;

  theFrontWindow = ( WindowPeek )FrontWindow();

  if ( thePWindow  == theFrontWindow  ||
    theFrontWindow ==
      NextVisWindow( anyKind, thePWindow ))
    return(( WindowPeek )nil );

  prevWindow = theFrontWindow;
  loopWindow = prevWindow->nextWindow;
  done = false;

  while ( done == false )
  {
    if ( loopWindow == thePWindow )
      done = true;
    else
      if ( loopWindow->visible != false )
        prevWindow = loopWindow;
      loopWindow = loopWindow->nextWindow;
  }

  return( prevWindow );
}
/*
 *  B R I N G F O R W A R D
 */
static void
BringForward( thePWindow, behindWindow,
              waitWithCalc )
  WindowPeek  thePWindow;
  WindowPeek  behindWindow;
  Boolean     waitWithCalc;
{
  CopyRgn  ( thePWindow->strucRgn, firstRgn );
  OffsetRgn( thePWindow->port.visRgn,
        - thePWindow->port.portBits.bounds.left,
        - thePWindow->port.portBits.bounds.top );
  DiffRgn  ( firstRgn, thePWindow->port.visRgn,
             firstRgn );
  OffsetRgn( thePWindow->port.visRgn,
          thePWindow->port.portBits.bounds.left,
          thePWindow->port.portBits.bounds.top );

  SendBehind(( WindowPtr )thePWindow,
             ( WindowPtr )behindWindow );
  PaintOne( thePWindow, firstRgn );
  if ( waitWithCalc == false )
    CalcVisBehind( thePWindow,
                   thePWindow->strucRgn );
  SetEmptyRgn( firstRgn );
}
/*
 *  H I L I T E T O O L S
 */

static void
HiliteTools( hilite )
  Boolean     hilite;
{
```


© The Definitive MacTutor, Vol. 4

```
  WindowPeek  loopWindow;
  Boolean     done;

  if ( frontToolWindow != nil )
  {
    for ( done = false,
            loopWindow = frontToolWindow;
          done == false  &&  loopWindow != nil;
          loopWindow = loopWindow->nextWindow )
    {
      if ( TGetWKind(( WindowPtr )loopWindow )
            == toolKind  &&
           loopWindow->visible != false )
      {
        HiliteWindow(( WindowPtr )loopWindow, hilite );
        if ( loopWindow == backToolWindow )
          done = true;
      }
    }
  }
}
```

Listing: TWindow.h

```
/*
 *  TWindows.h
 *
 *  Include file for usage with the TWindow
 *  Manager, an extended window manager that
 *  supports tool windows;  these are windows
 *  that always float on top, for palettes and
 *  tools.
 *
 *  Written in MPW C 2.0
 *
 *  Copyright Thomas Fruin 1988
 *  All rights reserved.
 */


/*
 *  T Y P E S
 */

  /* Define the standard toolbox types INTEGER
     and LONGINT in terms of the equivalents for
     the MPW C compiler. */

#define INTEGER       short
#define LONGINT       long


  /* Constants for the kindes of windows */

/*      dialogKind      2   */
/*      userKind        8   */
#define systemKind     -1
#define toolKind    30000
#define anyKind     30001
#define inFront   ((WindowPtr)-1)


  /* Callable functions in the TWindow Manager */

void          TInitWindows();
WindowPtr     TNewWindow();
WindowPtr     TGetNewWindow();
void          TCloseWindow();
void          TDisposeWindow();
void          TSelectWindow();
void          THideWindow();
void          TShowWindow();
WindowPtr     TFrontWindow();
void          TDragWindow();
Boolean       TGetNextEvent();
INTEGER       TGetWKind();
```

Listing: make TWindowTester

```
#
#   Makefile for TWindowTester and TWindows
#
#   TWindowTester is a simple program to demon-
#   strate the usage of tool windows by using
#   routines from TWindows,  a complete manager
#   for handling these windows that always float
#   on top, ususally used for palettes and tools.
#
#   Copyright Thomas Fruin 1988
#   All rights reserved
#

Libs    = "{CLibraries}"CInterface.o      ∂
          "{CLibraries}"CRuntime.o        ∂
          "{CLibraries}"glueenvirons.a.o  ∂
          "{CLibraries}"StdCLib.o

TWindowTester ƒƒ  TWindowTester.c.o TWindows.c.o
  Link TWindowTester.c.o  ∂
     TWindows.c.o         ∂
     {Libs}               ∂
     -o  TWindowTester    ∂
     -t  'APPL'           ∂
     -c  'TWIN'

TWindowTester ƒƒ  TWindowTester.r
  rez -o TWindowTester    ∂
     Types.r              ∂
     TWindowTester.r      ∂
     -a

TWindowTester.c.o ƒ TWindowTester.c TWindows.h
  C TWindowTester.c

TWindows.c.o      ƒ TWindows.c
  C TWindows.c -d __ALLNU__
```

Listing: TWindowTester.c

```
/*
 *TWindowTester
 *
 *C source of a program to demo usage of tool windows by
 *using routines from the TWindow Manager.  Tool windows  are
 *windows that always float on top, typically for palettes and
 *tools.
 *   Thomas Fruin 1988
 *
 *   fruin@hleru15.BITNET     University of Leiden
 *   thomas@uvabick.UUCP      University of Amsterdam
 *   dibs@well.UUCP
 *   hol0066.AppleLink
 *   2:508/15.FidoNet         The Netherlands
 *
 *TWindowTester is based on MiniEdit - Mini text
 * editor,converted from
 *the listing in Macintosh Revealed, vol II. As little as
 * possible was
 *modified in the original program, although most of the
 *functionality is no longer there.
 */

#include <Types.h>
#include <QuickDraw.h>
#include <Fonts.h>
#include <Windows.h>
#include <Events.h>
#include <TextEdit.h>
```

```
#include <Controls.h>
#include <Dialogs.h>
#include <Menus.h>
#include <Memory.h>
#include <OSUtils.h>
#include <ToolUtils.h>
#include <Desk.h>

#include <TWindows.h>

#define MenuBarHeight 20
#define TitleBarHeight 18
#define ScreenMargin 4

#define MinWidth    80
#define MinHeight   80
#define SBarWidth   16

#define AppleID      1
#define AboutItem    1

#define FileID       2
#define NewItem      1
#define  NewToolItem   2
#define CloseItem    3
#define QuitItem     5

#define EditID       3
#define UndoItem     1
#define CutItem      3
#define CopyItem     4
#define PasteItem    5
#define ClearItem    7

#define  WindowsID     4

#define  ToolsID       5

#define AboutID      128
#define windowID    1000
#define scrollID    1000
#define  toolID     1001

#define  okButton    1
#define  aboutBold     8

#define wHOffset    20
#define wVOffset    20
#define tHOffset    40
#define tVOffset    40

#define undoCmd      0
#define cutCmd       2
#define copyCmd      3
#define pasteCmd     4
#define clearCmd     5

  /* z-ordered list of windows (nearest first) */

#define windowList    (*(WindowPeek *)0x9D6)
#define  nilproc      ((ProcPtr)0)

struct WindowData
{
  ControlHandle scBar;
};

typedef struct WindowData  WindowData;
typedef struct WindowData  *WDPtr;
typedef struct WindowData  **WDHandle;

MenuHandle   AppleMenu,
        FileMenu,
        EditMenu,
        WindowsMenu,
        ToolsMenu;

INTEGER      toolCount,
        windowCount,
        lastToolsItem,
        lastWindowsItem;

Boolean      Finished,
        ErrorFlag;

main()
{
  Initialize();

  do
  {
    if ( FrontWindow() == nil )
      DisableItem( FileMenu, CloseItem );
    SystemTask();
    DoEvent();
  }
  while ( !Finished );
}

Initialize()
{
  INTEGER    theMask;

  InitGraf( &qd.thePort );
  InitFonts();
  InitWindows();
  InitMenus();
  TEInit();
  InitDialogs( nilproc );

  TInitWindows();

  theMask = everyEvent - keyUpMask;
  SetEventMask( theMask );
  FlushEvents( everyEvent, 0 );

  SetUpMenus();
  InitCursor();
}

SetUpMenus()
{
  AppleMenu = GetMenu( AppleID );
  AddResMenu( AppleMenu, 'DRVR' );
  InsertMenu( AppleMenu, 0 );

  FileMenu = GetMenu( FileID );
  InsertMenu( FileMenu, 0 );

  EditMenu = GetMenu( EditID );
  InsertMenu( EditMenu, 0 );

  WindowsMenu = GetMenu( WindowsID );
  InsertMenu( WindowsMenu, 0 );

  ToolsMenu = GetMenu( ToolsID );
  InsertMenu( ToolsMenu, 0 );

  DrawMenuBar();

  lastToolsItem   =
  lastWindowsItem = 0;
}

DoEvent()
{
  EventRecordtheEvent;
```


© The Definitive MacTutor, Vol. 4

```
  ErrorFlag = false;

  if ( TGetNextEvent( everyEvent, &theEvent ))
  {
    switch ( theEvent.what )
    {
      case mouseDown:
        DoMouseDown( &theEvent );
        break;

      case keyDown:
      case autoKey:
        DoKeyStroke( &theEvent );
        break;

      case updateEvt:
        DoUpdate( &theEvent );
        break;

      case activateEvt:
        DoActivate( &theEvent );
        break;

      default:
        break;
    }
  }
}

DoMouseDown( theEvent )
  EventRecord*theEvent;
{
  WindowPtr  whichWindow;
  INTEGER    thePart;

  thePart = FindWindow( &theEvent->where, &whichWindow );

  switch ( thePart )
  {
    case inDesk:
      break;

    case inMenuBar:
      DoMenuClick( theEvent );
      break;

    case inSysWindow:
      SystemClick( theEvent, whichWindow );
      break;

    case inContent:
      DoContent( theEvent, whichWindow );
      break;

    case inDrag:
      DoDrag( theEvent, whichWindow );
      break;

    case inGrow:
      DoGrow( theEvent, whichWindow );
      break;

    case inGoAway:
      DoGoAway( theEvent, whichWindow );
      break;
  }
}

DoMenuClick( theEvent )
  EventRecord*theEvent;
{
  LONGINT  menuChoice;

  menuChoice = MenuSelect( &theEvent->where );
  DoMenuChoice( menuChoice );
}

DoMenuChoice( menuChoice )
  LONGINT    menuChoice;
{
  INTEGER    theMenu,
             theItem;

  if ( menuChoice )
  {
    theMenu = HiWord( menuChoice );
    theItem = LoWord( menuChoice );

    switch (theMenu)
    {
      case AppleID:
        DoAppleChoice( theItem );
        break;

      case FileID:
        DoFileChoice( theItem );
        break;

      case EditID:
        DoEditChoice( theItem );
        break;

      case WindowsID:
        DoWindowsChoice( WindowsMenu, theItem );
        break;

      case ToolsID:
        DoWindowsChoice( ToolsMenu, theItem );
        break;
    }
    HiliteMenu( 0 );
  }
}

DoAppleChoice( theItem )
  INTEGER    theItem;
{
  char   accName[255];
  INTEGER    accNumber;

  switch ( theItem )
  {
    case AboutItem:
      DoAbout();
      break;

    default:
      EnableItem( FileMenu, CloseItem );

      EnableItem( EditMenu, UndoItem  );
      EnableItem( EditMenu, CutItem   );
      EnableItem( EditMenu, CopyItem  );
      EnableItem( EditMenu, PasteItem );
      EnableItem( EditMenu, ClearItem );

      GetItem( AppleMenu, theItem, accName );
      accNumber = OpenDeskAcc( accName );
      break;
  }
}

DoAbout()
{
  EventRecordtheEvent;
  DialogPtr  aboutDialog;
  INTEGER    theItem;
  void     SetBold();
```

```
  aboutDialog = GetNewDialog( AboutID, nil, inFront );
  SetBold( aboutDialog, aboutBold );
  TShowWindow( aboutDialog );

  if ( TGetNextEvent( activMask, &theEvent ))
    DoActivate( &theEvent );

  if ( TGetNextEvent( activMask, &theEvent ))
    DoActivate( &theEvent );

  ModalDialog( nilproc, &theItem );

  THideWindow ( aboutDialog );
  DisposDialog( aboutDialog );
}

DoFileChoice( theItem )
  INTEGER    theItem;
{
  switch (theItem)
  {
    case NewItem:
      DoNew();
      break;

    case NewToolItem:
      DoNewTool();
      break;

    case CloseItem:
      DoClose();
      break;

    case QuitItem:
      DoQuit();
      break;
  }
}

DoNew()
{
  WindowPtr  theWindow;
  WDHandle theData;
  char   title[ 255 ];

  theWindow = TGetNewWindow(userKind, windowID, nil,inFront );

  OffsetWindow( theWindow, wHOffset, wVOffset, &windowCount );
  TShowWindow ( theWindow );

  SetPort( theWindow );

  theData = ( WDHandle )NewHandle(( Size )sizeof( WindowData
));
  SetWRefCon( theWindow, ( LONGINT )theData );
  HLock( theData );

  ( *theData )->scBar = GetNewControl( scrollID, theWindow );

  HUnlock( theData );

  EnableItem( FileMenu, CloseItem );

  GetWTitle( theWindow, title );
  InsMenuItem( WindowsMenu, title, lastWindowsItem++ );
  CheckItem  ( WindowsMenu, lastWindowsItem, true );
}

DoNewTool()
{
  WindowPtr  toolWindow;
  char   title[ 255 ];

  toolWindow = TGetNewWindow( toolKind, toolID, nil, inFront
);

  OffsetWindow( toolWindow, tHOffset, tVOffset, &toolCount );
  TShowWindow( toolWindow );

  SetPort( toolWindow );

  EnableItem( FileMenu, CloseItem );

  GetWTitle( toolWindow, title );
  InsMenuItem( ToolsMenu, title, lastToolsItem++ );
  CheckItem  ( ToolsMenu, lastToolsItem, true );
}

OffsetWindow( whichWindow, hOffset, vOffset, count )
  WindowPtr  whichWindow;
  INTEGER    hOffset,
         vOffset,
         *count;
{
  INTEGER    windowWidth,
         windowHeight,
         hExtra,
         vExtra,
         hMax,
         vMax,
         windowLeft,
         windowTop;
  Rect   pRect;
  char   title [ 255 ],
         number[ 3 ];

  pRect = whichWindow->portRect;

  windowWidth   = pRect.right - pRect.left;
  windowHeight  = pRect.bottom - pRect.top;
  windowHeight += TitleBarHeight;

  hExtra = qd.screenBits.bounds.right  -   windowWidth;
  vExtra = qd.screenBits.bounds.bottom - ( windowHeight +
MenuBarHeight );

  hMax = ( hExtra / hOffset ) + 1;
  vMax = ( vExtra / vOffset ) + 1;

  ++*count;

  windowLeft = ( *count % hMax ) * hOffset;
  windowTop  = ( *count % vMax ) * vOffset;
  windowTop  = windowTop + TitleBarHeight + MenuBarHeight;

  MoveWindow( whichWindow, windowLeft, windowTop, false );

  GetWTitle( whichWindow, title );
  NumToString(( LONGINT )*count, number );
  SetWTitle( whichWindow, strcat( title, number ));
}

DoClose()
{
  WindowPtr  whichWindow;
  INTEGER    theWKind;

  whichWindow = TFrontWindow( userKind );

  if ( whichWindow != nil )
  {
    theWKind = TGetWKind( whichWindow );
    switch( theWKind )
    {
      case userKind:
          CloseDocWindow( whichWindow );
          break;
```


© The Definitive MacTutor, Vol. 4

```
      case systemKind:
          CloseSysWindow ( whichWindow );
          break;
    }
  }
  else
  {
    whichWindow = TFrontWindow( toolKind );

    if ( whichWindow != nil )
      CloseToolWindow( whichWindow );
  }
}

CloseDocWindow( whichWindow )
  WindowPtr  whichWindow;
{
  WDHandle theData;
  INTEGER    ItemOf();
  EventRecordtheEvent;

  lastWindowsItem-;
  DelMenuItem( WindowsMenu, ItemOf( whichWindow, WindowsMenu
));

  theData = ( WDHandle )GetWRefCon( whichWindow );

  THideWindow( whichWindow );

  if ( TGetNextEvent( activMask, &theEvent ))
    DoActivate( &theEvent );

  if ( TGetNextEvent( activMask, &theEvent ))
    DoActivate( &theEvent );

  if ( theData != nil )
    DisposHandle( theData );
  TDisposeWindow( whichWindow );
}

CloseToolWindow( whichWindow )
  WindowPtr  whichWindow;
{
  INTEGER    ItemOf();

  lastToolsItem-;
  DelMenuItem( ToolsMenu, ItemOf( whichWindow, ToolsMenu ));
  TDisposeWindow( whichWindow );
}

CloseSysWindow( whichWindow )
  WindowPtr  whichWindow;
{
  INTEGER    accNumber;

  accNumber = (( WindowPeek )whichWindow )->windowKind;
  CloseDeskAcc( accNumber );
}

DoQuit()
{
  Finished = true;
}

DoEditChoice( theItem )
  INTEGER    theItem;
{
  switch ( theItem )
  {
    case UndoItem:
      if ( !SystemEdit( undoCmd  ))
        DoUndo();
      break;

    case CutItem:
      if ( !SystemEdit( cutCmd   ))
        DoCut();
      break;

    case CopyItem:
      if ( !SystemEdit( copyCmd  ))
        DoCopy();
      break;

    case PasteItem:
      if ( !SystemEdit( pasteCmd ))
        DoPaste();
      break;

    case ClearItem:
      if ( !SystemEdit(clearCmd  ))
        DoClear();
      break;
  }
}

DoUndo()
{
  SysBeep(1);
}

DoCut()
{
  SysBeep(1);
}

DoCopy()
{
  SysBeep(1);
}

DoPaste()
{
  SysBeep(1);
}

DoClear()
{
  SysBeep(1);
}

DoWindowsChoice( theMenu, theItem )
  MenuHandle theMenu;
  INTEGER    theItem;
{
  INTEGER    theMark;
  WindowPtr  whichWindow,
         WindowOf();

  whichWindow = WindowOf( theMenu, theItem );
  GetItemMark( theMenu, theItem, &theMark );

  if ( theMark == noMark )
  {
    CheckItem( theMenu, theItem, true );
    TShowWindow( whichWindow );
  }
  else
  {
    CheckItem( theMenu, theItem, false );
    THideWindow( whichWindow );
  }
}

DoContent( theEvent, whichWindow )
  EventRecord*theEvent;
  WindowPtr  whichWindow;
{
```

```
  INTEGER    theWKind;

  theWKind = TGetWKind( whichWindow );

  if ( theWKind == userKind )
  {
    if ( whichWindow != TFrontWindow( userKind ))
      TSelectWindow( whichWindow );
  }
  else if ( theWKind == toolKind )
  {
    if ( whichWindow != TFrontWindow( toolKind ))
      TSelectWindow( whichWindow );

    /* Only if the window has no title bar */

    DoDrag( theEvent, whichWindow );
  }
}

DoDrag( theEvent, whichWindow)
  EventRecord*theEvent;
  WindowPtr  whichWindow;
{
  Rect   limitRect;

  SetRect( &limitRect, 0, MenuBarHeight,
           qd.screenBits.bounds.right,
           qd.screenBits.bounds.bottom );
  InsetRect( &limitRect, ScreenMargin, ScreenMargin );
  TDragWindow( whichWindow, theEvent, &limitRect );
}

DoGrow( theEvent, whichWindow )
  EventRecord*theEvent;
  WindowPtr  whichWindow;
{
  Rect   sizeRect;
  LONGINT    newSize;
  INTEGER    newWidth,
        newHeight;
  GrafPtr    savePort;

  if ( whichWindow != TFrontWindow( userKind ))
    TSelectWindow( whichWindow );
  else
  {
    SetRect( &sizeRect, MinWidth, MinHeight,
             qd.screenBits.bounds.right,
             qd.screenBits.bounds.bottom - MenuBarHeight );
    newSize = GrowWindow( whichWindow, &theEvent->where,
&sizeRect );

    if ( newSize )
    {
      GetPort( &savePort );
      SetPort( whichWindow );
      EraseRect( &whichWindow->portRect );
      newWidth  = LoWord( newSize );
      newHeight = HiWord( newSize );
      SizeWindow( whichWindow, newWidth, newHeight, true );
      InvalRect( &whichWindow->portRect );
      FixScrollBar( whichWindow );
      SetPort( savePort );
    }
  }
}

FixScrollBar( whichWindow )
  WindowPtr    whichWindow;
{
  WDHandle   theData;
  ControlHandle theScrollBar;
  Rect     pRect;

  theData = ( WDHandle )GetWRefCon( whichWindow );
  if ( theData != nil )
  {
    theScrollBar = ( *theData )->scBar;
    HideControl( theScrollBar );
    pRect = whichWindow->portRect;

    MoveControl( theScrollBar, pRect.right - ( SBarWidth - 1
), -1 );
    SizeControl( theScrollBar, SBarWidth, ( pRect.bottom + 1 )
          - ( pRect.top - 1 ) - ( SBarWidth - 1 ));
    ShowControl( theScrollBar );
    ValidRect( &( *theScrollBar )->contrlRect );
  }
}

DoGoAway( theEvent, whichWindow )
  EventRecord*theEvent;
  WindowPtr  whichWindow;
{
  INTEGER    theWKind;

  theWKind = TGetWKind( whichWindow );

  if ( theWKind == userKind )
  {
    if ( whichWindow != TFrontWindow( userKind ))
      TSelectWindow( whichWindow );
    else if ( TrackGoAway( whichWindow, &theEvent->where ))
      CloseDocWindow( whichWindow );
  }
  else if ( theWKind == toolKind )
  {
    if ( TrackGoAway( whichWindow, &theEvent->where ))
    {
      lastToolsItem-;
      DelMenuItem( ToolsMenu, ItemOf( whichWindow, ToolsMenu
));
      TDisposeWindow( whichWindow );
    }
  }
}

DoKeyStroke( theEvent )
  EventRecord*theEvent;
{
  INTEGER  ch;
  LONGINT  menuChoice;

  ch = ( INTEGER )( theEvent->message & charCodeMask );

  if ( theEvent->modifiers & cmdKey )
  {
    if ( theEvent->what != autoKey )
    {
      menuChoice = MenuKey( ch );
      DoMenuChoice( menuChoice );
    }
  }
}

DoUpdate( theEvent )
  EventRecord*theEvent;
{
  GrafPtr    savePort;
  WindowPtr  whichWindow;
  WDHandle theData;
  INTEGER    theWKind;

  GetPort( &savePort );

  whichWindow = ( WindowPtr )theEvent->message;
  theWKind = TGetWKind( whichWindow );
```


© The Definitive MacTutor, Vol. 4

```
  SetPort( whichWindow );
  BeginUpdate( whichWindow );

  if ( theWKind == userKind )
  {
    EraseRect( &whichWindow->portRect );

    if ( GetWRefCon( whichWindow ) != nil )
    {
      DrawGrowIcon( whichWindow );
      DrawControls( whichWindow );
    }
  }
  else if ( theWKind == toolKind )
  {
    EraseRect( &whichWindow->portRect );
  }
  EndUpdate( whichWindow );
  SetPort( savePort );
}

DoActivate( theEvent )
  EventRecord*theEvent;
{
  WindowPtr  whichWindow,
        theFrontWindow;
  WDHandle theData;
  INTEGER    theWKind,
        frontWKind;
  whichWindow = ( WindowPtr )theEvent->message;
  SetPort( whichWindow );
  theWKind = TGetWKind( whichWindow );
  if ( theWKind == userKind )
  {
    theData = (WDHandle )GetWRefCon( whichWindow );
    if ( theData != nil )
    {
      HLock( theData );
      if ( theEvent->modifiers & activeFlag )
      {
        ShowControl(( *theData )->scBar );

        DisableItem( EditMenu, UndoItem  );
        DisableItem( EditMenu, CutItem   );
        DisableItem( EditMenu, CopyItem  );
        DisableItem( EditMenu, PasteItem );
        DisableItem( EditMenu, ClearItem );
      }
      else
      {
        HideControl(( *theData )->scBar );

        theFrontWindow = FrontWindow();
        frontWKind     = TGetWKind( theFrontWindow );
        if ( frontWKind == systemKind )
        {
          EnableItem( EditMenu, UndoItem  );
          EnableItem( EditMenu, CutItem   );
          EnableItem( EditMenu, CopyItem  );
          EnableItem( EditMenu, PasteItem );
          EnableItem( EditMenu, ClearItem );
        }
      }
      HUnlock( theData );
      DrawGrowIcon( whichWindow );
    }
  }
}

INTEGER
ItemOf( whichWindow, whichMenu )
  WindowPtr  whichWindow;
  MenuHandle whichMenu;
{
  char   theTitle[ 255 ],
         theMenu [ 255 ];
  Boolean    done;
  INTEGER    theItem;

  GetWTitle( whichWindow, theTitle );

  for ( done = false,
      theItem = 1;
      done == false;
      theItem++    )
  {
    GetItem( whichMenu, theItem, theMenu );
    if ( strcmp( theTitle, theMenu ) == 0 )
    {
      done = true;
      theItem-;
    }
  }

  return( theItem );
}

WindowPtr
WindowOf( whichMenu, whichItem )
  MenuHandle whichMenu;
  INTEGER    whichItem;
{
  char   theTitle[ 255 ],
         theMenu [ 255 ];
  Boolean    done;
  INTEGER    wantedKind,
         theWKind;
  WindowPeek loopWindow,
         wantedWindow;

  GetItem( whichMenu, whichItem, theMenu );
  if ( whichMenu == WindowsMenu )
    wantedKind = userKind;
  else
    wantedKind = toolKind;

  for ( done = false,
      loopWindow   =
      wantedWindow = windowList;
      done == false  &&  loopWindow != nil;
      loopWindow = loopWindow->nextWindow )
  {
    theWKind = TGetWKind( loopWindow );
    if ( theWKind == wantedKind )
    {
      GetWTitle( loopWindow, theTitle );
      if ( strcmp( theTitle, theMenu ) == 0 )
      {
        wantedWindow = loopWindow;
        done = true;
      }
    }
  }

  return(( WindowPtr )wantedWindow );
}

void
SetBold( theDialog, bolditem )
  DialogPtr    theDialog;
  short      bolditem;
{
  INTEGER      itemType;
  Handle     itemHandle;
  Rect     itemRect;
  pascal void  OutlineButton();

    /* Get the OK button's display rectangle, enlarge the
```

```
      rectangle, and set up the useritem with this rectangle
      and a pointer to the OutlineButton function. */

  GetDItem( theDialog, okButton, &itemType, &itemHandle,
&itemRect );
  InsetRect( &itemRect, -3, -3 );
  SetDItem( theDialog, bolditem, userItem,
       ( Handle )OutlineButton, &itemRect );
}

pascal void
OutlineButton( theDialog, theItem )
  DialogPtr    theDialog;
  INTEGER      theItem;
{
  INTEGER      itemType;
  Handle     itemHandle;
  Rect     itemRect;
  PenState   savePen;

    /* Get the bounding rectangle of the OK button item
       and boldly outline it. */

  GetDItem( theDialog, okButton, &itemType, &itemHandle,
&itemRect );
  GetPenState( &savePen );
  PenSize( 3,3 );
  InsetRect( &itemRect, -4, -4 );
  FrameRoundRect( &itemRect, 16, 16 );
  SetPenState( &savePen );
}
```

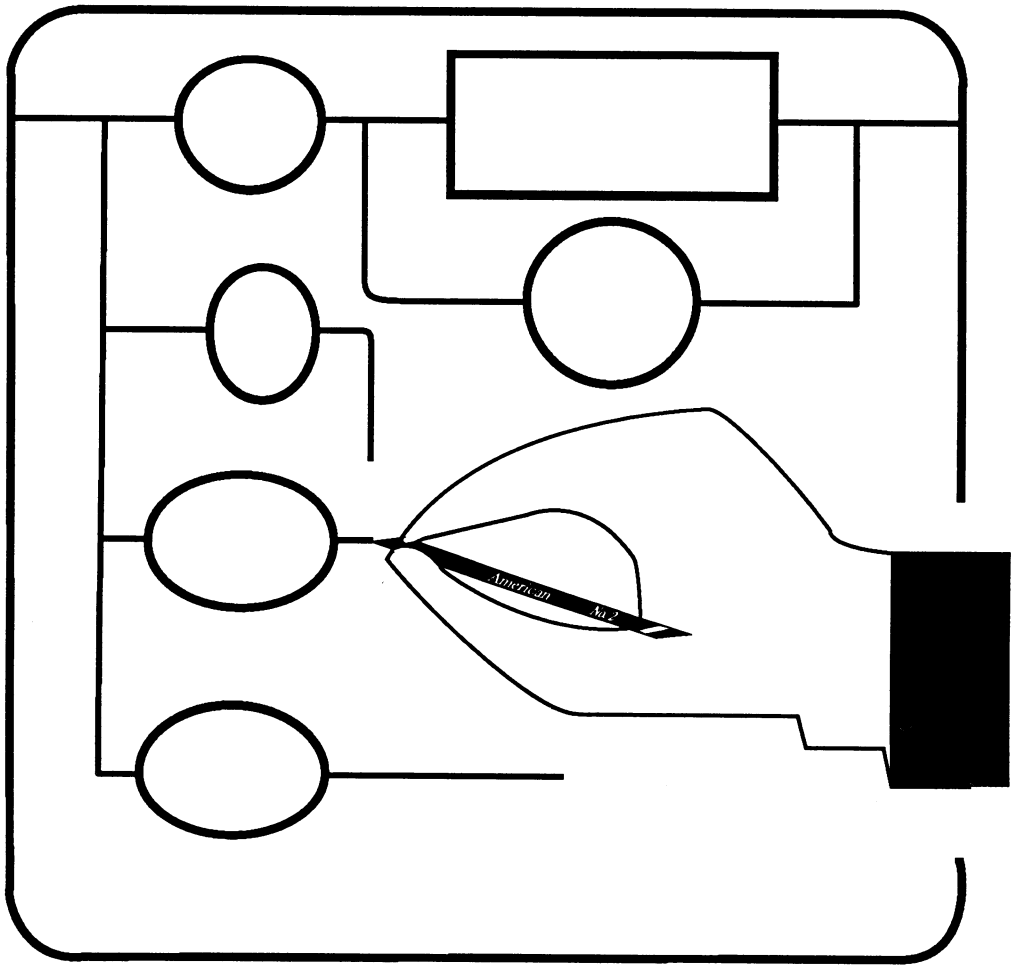
Listing: TWindowTester.R

```
/*
 *TWindowTester.r
 *
 *Resources for a simple program to demonstrate usage of tool
windows
 *by using routines from the TWindow Manager. Tool windows are
windows
 *that always float on top, typically for palettes and tools.
 *
 *Written in MPW C 2.0
 *
 *  Thomas Fruin 1988
 *
 *  fruin@hleru15.BITNET    University of Leiden
 *  thomas@uvabick.UUCP       University of Amsterdam
 *  dibs@well.UUCP
 *  hol0066.AppleLink
 *  2:508/15.FidoNet        The Netherlands
 *
 *TWindowTester is based on MiniEdit - Mini text
editor,converted from
 *the listing in Macintosh Revealed, vol II. As little as
possible was
 *modified in the original program, although most of the
functionality
 *is no longer there.
 */

resource 'MENU' (1) {
  1,
  textMenuProc,
  0x7FFFFFFD,
  enabled,
  apple,
  { /* array: 2 elements */
    /* [1] */
    "About TWindowTester ...", noIcon, "", "", plain,
    /* [2] */
    "-", noIcon, "", "", plain
  }
};

resource 'MENU' (2) {
  2,
  textMenuProc,
  0x7FFFFFF3,
  enabled,
  "File",
  { /* array: 5 elements */
    /* [1] */
    "New", noIcon, "N", "", plain,
    /* [2] */
    "New Tool", noIcon, "T", "", plain,
    /* [3] */
    "Close", noIcon, "W", "", plain,
    /* [4] */
    "-", noIcon, "", "", plain,
    /* [5] */
    "Quit", noIcon, "Q", "", plain
  }
};

resource 'MENU' (3) {
  3,
  textMenuProc,
  0x7FFFFF80,
  enabled,
  "Edit",
  { /* array: 7 elements */
    /* [1] */
    "Undo", noIcon, "Z", "", plain,
    /* [2] */
    "-", noIcon, "", "", plain,
    /* [3] */
    "Cut", noIcon, "X", "", plain,
    /* [4] */
    "Copy", noIcon, "C", "", plain,
    /* [5] */
    "Paste", noIcon, "V", "", plain,
    /* [6] */
    "-", noIcon, "", "", plain,
    /* [7] */
    "Clear", noIcon, "B", "", plain
  }
};

resource 'MENU' (4) {
  4,
  textMenuProc,
  allEnabled,
  enabled,
  "Windows",
  { /* array: 0 elements */
  }
};

resource 'MENU' (5) {
  5,
  textMenuProc,
  allEnabled,
  enabled,
  "Tools",
  { /* array: 0 elements */
  }
};

resource 'WIND' (1000) {
  {50, 40, 300, 450},
  documentProc,
  invisible,
  goAway,
  0x0,
  "Document "
};
```


© The Definitive MacTutor, Vol. 4

```
resource 'WIND' (1001) {
  (58, 533, 159, 606),
  altDBoxProc,
  invisible,
  noGoAway,
  0x0,
  "Tool "
};

resource 'CNTL' (1000) {
  (-1, 395, 236, 411),
  0,
  -1,
  0,
  0,
  scrollBarProc,
  0,
  "vertical scroll bar"
};

resource 'PICT' (128) {
  2616,
  (0, 5, 217, 188),
  $"1101 A000 82A0 3039 A000 8E01 000A 0000"
  $"0000 02D0 0240 9800 1800 0000 0000 8600"
  $"C000 0000 0500 8600 BC00 0000 0500 8600"
  $"BC00 0106 FC00 0030 EF00 06FC 0000 78EF"
  $"0006 FC00 0078 EF00 06FC 0000 32EF 0006"
  $"FC00 0007 EF00 06FC 0000 0EEF 0007 FC00"
  $"0104 C0F0 000A 0101 80FE 0001 01E0 F000"
  $"0A01 03C0 FE00 0101 E0F0 0009 0107 E0FD"
  $"0000 C8F0 0009 0107 E0FD 0000 1CF0 0009"
  $"0107 E0FD 0000 38F0 0009 0103 C4FD 0000"
  $"13F0 000A 0101 8EFD 0001 0780 F100 0A01"
  $"001E FD00 0107 80F1 000B 0700 3C00 0060"
  $"0003 20F1 000B 0700 38C0 00F0 0000 70F1"
  $"000B 0700 11E0 00F0 0000 E8F1 000B 0700"
  $"03F0 0060 0000 5CF1 000B 0700 03F0 000F"
  $"F000 38F1 000B 0700 03F0 003F FC00 13F1"
  $"000C 0800 01E0 00FF FF00 0780 F200 0C08"
  $"0000 C301 FFFF 8007 80F2 000B 0700 0007"
  $"83FF FFC0 03F1 000A 0600 000F C7FF FFE0"
  $"F000 0A06 0000 0FCF FFFF F0F0 000A 0600"
  $"000F CFFE FFF0 F000 0A06 0000 079F FE7F"
  $"F8F0 000A 0600 0003 1FFE 3FF8 F000 09FE"
  $"0003 3FFE 1FFC F000 09FE 0003 3FFE 3FFC"
  $"F000 09FE 0003 3FFC 7FFC F000 09FE 0003"
  $"3FFC 7BFC F000 09FE 0003 3FF8 F9FC F000"
  $"09FE 0003 3FF8 F8FC F000 09FE 0003 3FF9"
  $"E3FC F000 09FE 0003 3FF1 0FFC F000 09FE"
  $"0003 3FF0 7FFC F000 0AFE 0004 3FE1 FFFC"
  $"C0F1 000A FE00 041F F7FF F9E0 F100 0AFE"
  $"0004 1FFF FFF9 E0F1 000A FE00 040F FFFF"
  $"F0C8 F100 0AFE 0004 0FFF FFF0 1CF1 000A"
  $"FE00 0407 FFFF E038 F100 0AFE 0004 03FF"
  $"FFC0 13F1 000B FE00 0501 FFFF 8007 80F2"
  $"000A FD00 04FF FF00 0780 F200 09FD 0003"
  $"3FFC 0003 F100 07FD 0001 0FF0 EF00 02E9"
  $"0009 FB00 0330 000F F0F3 0009 FB00 0378"
  $"003F FCF3 0009 FB00 03F0 00FF FFF3 000B"
  $"FC00 0501 E201 FFFF 80F4 000B FC00 0501"
  $"C703 FFFF C0F4 000A FB00 048F 07FF FFE0"
  $"F400 0AFB 0004 1E0F FFFF F0F4 000A FB00"
  $"043C 0FFF FFF0 F400 0AFB 0004 181F FFFF"
  $"F8F4 0009 FA00 031F FFFF F8F4 0009 FA00"
  $"033F FFEF FCF4 0009 FA00 033F FF87 FCF4"
  $"0009 FA00 033F FC0F FCF4 000B FC00 0530"
  $"003F F08F FCF4 000B FC00 0578 003E 038F"
  $"FCF4 000B FC00 0578 003F 391F FCF4 000B"
  $"FC00 0532 003F BC1F FCF4 000B FC00 0507"
  $"003F FE1F FCF4 000B FC00 050E 003F FF3F"
  $"FCF4 000B FC00 0504 C03F FE3F FCF4 000C"
  $"FC00 0601 E01F F83F F8C0 F500 0CFC 0006"
  $"01E0 1FFC 7FF9 E0F5 000B FB00 05D8 0FFE"
  $"7FF1 E0F5 000B FB00 053C 0FFF 7FF0 C8F5"
  $"000B FB00 053C 07FF FFE0 1CF5 000B FB00"
  $"051B 03FF FFC0 3AF5 000B FB00 0507 81FF"
  $"FF80 17F5 000B FB00 0507 80FF FF00 0EF5"
  $"000C FB00 0603 603F FC00 04C0 F600 0BFA"
  $"0005 F00F F000 01E0 F600 0BFA 0000 F0FE"
  $"0001 01E0 F600 0CFA 0006 6C00 0020 00CF"
  $"F0F7 000C FA00 061E 0000 7000 3FFC F700"
  $"0CFA 0006 1E00 00F0 00FF FFF7 000D FA00"
  $"070C 8001 E001 FFFF 80F8 000D FA00 0701"
  $"C001 C603 FFFF C0F8 000D FA00 0703 8000"
  $"8F07 FFFF E0F8 000D FA00 0701 0000 1F8F"
  $"FDFF F0F8 000A F700 041F 8FFC FFF0 F800"
  $"0AF7 0004 1F9F F87F F8F8 000A F700 040F"
  $"1FF0 7FF8 F800 0DF7 0007 063F E4FF FC00"
  $"0030 FB00 0CF6 0006 3FC8 FFFC 0000 78FB"
  $"000C F600 063F 99FF FC00 0078 FB00 0CF6"
  $"0006 3F39 FBFC 0000 32FB 000C F600 063C"
  $"71F1 FC00 0007 FB00 0CF6 0006 3E73 C0FC"
  $"0000 0EFB 000D F600 073F 7330 FC00 0004"
  $"C0FC 000D F600 073F F0E3 FC00 0001 E0FC"
  $"000D F600 073F E3C7 FC00 0001 E0FC 000F"
  $"F800 0530 003F EF8F FCFE 0000 C8FC 000F"
  $"F800 0578 001F FF1F F8FE 0000 1CFC 000F"
  $"F800 0978 001F F83F F840 0000 38FC 000F"
  $"F800 0932 000F FC7F F0E0 0000 13FC 0010"
  $"F800 0A07 000F FE7F F1D0 0000 0780 FD00"
  $"10F8 000A 0E00 07FF 7FE0 B800 0007 80FD"
  $"0010 F800 0A04 C003 FFFF C074 0000 0360"
  $"FD00 10F8 0006 01E0 01FF FF80 2EFE 0000"
  $"F0FD 0010 F800 0601 E000 FFFF 001C FE00"
  $"00F0 FD00 0FF7 0009 D800 3FFC 0009 8000"
  $"006C FD00 0FF7 0009 3C00 0FF0 0003 C000"
  $"001E FD00 0EF7 0000 3CFD 0004 03C0 0000"
  $"1EFD 0010 F700 0A19 0000 0180 0180 0000"
  $"0D80 FE00 10F7 000A 0380 0003 C000 1FE0"
  $"0003 C0FE 0010 F700 0A07 4000 07E0 007F"
  $"F800 03C0 FE00 10F7 000A 02E0 0007 E001"
  $"FFFE 0001 80FE 000D F700 0701 C000 07E0"
  $"03FF FFFB 000D F600 0798 0003 C407 FFFF"
  $"80FC 000F F600 093C 0001 8E0F FFFF C000"
  $"01FE 0010 F600 0A3C 0000 1E1F FFFF E000"
  $"0380 FF00 10F6 000A 1900 003C 1FFF FFE0"
  $"0006 C0FF 0010 F600 0A03 8000 383F FFFF"
  $"F000 0C60 FF00 10F6 000A 0700 0010 3FFE"
  $"FFF0 0018 30FF 0010 F600 0002 FE00 067F"
  $"FE7F F800 3018 FF00 0CF2 0006 7FFC 3FF8"
  $"0060 0CFF 000C F200 067F F81F F800 C006"
  $"FF00 0CF2 0006 7FF0 6FF8 00C0 03FF 000C"
  $"F200 087F E0EF F800 E001 8000 0CF2 0008"
  $"7FC1 E7F8 00B0 00C0 000C F200 087F 81C7"
  $"F800 9800 6000 0CF2 0008 7E04 07F8 018C"
  $"0030 000C F200 087F 0E0F F803 C600 1800"
  $"0CF2 0008 7F9F 9FF8 0663 000C 000C F200"
  $"083F DFFF F0CC 3180 0600 9800 1800 8600"
  $"0000 D900 C000 8600 0500 D900 BC00 8600"
  $"0500 D900 BC00 010C F200 083F FFFF F1F8"
  $"1980 0300 0CF2 0008 1FFF FFE3 300F 0001"
  $"800C F200 081F FFFF E618 0600 00C0 0CF2"
  $"0008 0FFF FFCC 0C0C 0000 600C F200 0807"
  $"FFFF 9806 1800 0030 0CF2 0008 03FF FF30"
  $"0330 0000 700C F200 0801 FFFE 7001 E000"
  $"00D0 0BF1 0007 7FF8 5800 C000 0190 0BF1"
  $"0002 1FE0 4CFE 0001 0330 09EF 0000 66FE"
  $"0001 0660 0AF0 0001 01B3 FE00 010C C00A"
  $"F000 0603 D980 0000 1980 0AF0 0006 07EC"
  $"C000 0033 000A F000 0607 E660 0000 6600"
  $"0AF0 0006 07E3 3000 00CC 000A F000 0603"
  $"C598 0001 9800 0AF0 0006 018E CC00 0330"
  $"0009 EF00 051E E600 0670 0009 EF00 053D"
  $"B300 0CD8 0009 EF00 053B F980 19CC 0009"
  $"EF00 0517 FCC0 33E6 0009 EF00 050F FE60"
  $"67F3 0009 EF00 051B F330 CFF9 8009 EF00"
```

```
  $"0533 F199 9FFC C009 EF00 0567 E0CF 3FFE"
  $"6009 EF00 05CC C7E6 7FFF 300A F000 0601"
  $"980F F0FF FF30 0AF0 0006 03CC 0FF9 FFFE"
  $"700A F000 0606 660F FFFF FCD0 0AF0 0006"
  $"0433 0FFF FFF9 900A F000 0604 1987 FFFF"
  $"F310 0AF0 0006 040C C3FF FFE6 100A F000"
  $"0604 0663 FFFF CCD0 0AF0 0006 0403 37FF"
  $"FF99 D00A F000 0604 019F FFFF 33D0 0AF0"
  $"0006 0400 CFFF FE67 D00A F000 0604 0067"
  $"FFFC CF50 0AF0 0006 0400 33FF F99A 500A"
  $"F000 0604 0019 FFF3 3250 0AF0 0006 0400"
  $"0CFF E662 500A F000 0604 0006 7FCC C250"
  $"0AF0 0006 0400 033F 9982 500A F000 0604"
  $"0001 9F33 0250 0AF0 0006 0400 00CE 6602"
  $"500A F000 0604 0000 64CC 0250 0AF0 0006"
  $"0400 0031 9806 500A F000 0604 0000 1B30"
  $"0CD0 0AF0 0006 0400 000E 6019 900A F000"
  $"0604 0000 04C0 3310 0AF0 0000 04FE 0002"
  $"8066 500A F000 0004 FE00 0280 CCD0 0AF0"
  $"0000 04FE 0002 8199 900A F000 0004 FE00"
  $"0283 3330 0AF0 0000 04FE 0002 8666 600A"
  $"F000 0004 FE00 028C CCC0 0AF0 0000 04FE"
  $"0002 9999 C00A F000 0004 FE00 02B3 3340"
  $"0AF0 0000 04FE 0002 E626 400A F000 0004"
  $"FE00 02CC 0C40 0AF0 0000 04FE 0002 9818"
  $"C00A F000 0004 FE00 02B0 3180 0AF0 0000"
  $"06FE 0002 E063 000A F000 0003 FE00 02C0"
  $"C600 0AF0 0006 0180 0000 818C 0009 EF00"
  $"05C0 0000 0318 0009 EF00 0560 0000 0630"
  $"0009 EF00 0530 0000 0C60 0009 EF00 0518"
  $"0000 18C0 0009 EF00 050C 0000 3180 0009"
  $"EF00 0306 0000 63FF 0009 EF00 0303 0020"
  $"C6FF 0009 EF00 0301 8031 8CFF 0008 EE00"
  $"02C0 1B18 FF00 08EE 0002 600E 30FF 0008"
  $"EE00 0230 0460 FF00 08EE 0002 1800 C0FF"
  $"0008 EE00 020C 0180 FF00 07EE 0001 0603"
  $"FE00 07EE 0001 0306 FE00 07EE 0001 018C"
  $"FE00 06ED 0000 D8FE 0006 ED00 0070 FE00"
  $"06ED 0000 20FE 00A0 008F A000 83FF"
};

resource 'DLOG' (128) {
  (62, 77, 312, 437),
  dBoxProc,
  invisible,
  noGoAway,
  0x0,
  128,
  "About"
};

resource 'DITL' (128) {
  { /* array DITLarray: 7 elements */
    /* [1] */
    (27, 270, 48, 341),
    Button {
      enabled,
      "OK"
    },
    /* [2] */
    (56, 66, 73, 174),
    StaticText {
      disabled,
      "TWindowTester"
    },
    /* [3] */
    (189, 19, 226, 266),
    StaticText {
      disabled,
      "Originally written for VAMP, "
      "the Dutch Mac programmers group."
    },
    /* [4] */
    (99, 60, 116, 172),
    StaticText {
      disabled,
      "by Thomas Fruin"
    },
    /* [5] */
    (117, 34, 135, 196),
    StaticText {
      disabled,
      "Leiden, The Netherlands"
    },
    /* [6] */
    (148, 68, 164, 169),
    StaticText {
      disabled,
      "20 April 1988"
    },
    /* [7] */
    (9, 158, 226, 341),
    Picture {
      disabled,
      128
    },
    /* [8] */
    (27, 270, 48, 341),
    UserItem {
      disabled
    }
  }
};
```



Pascal Procedures

© The Definitive MacTutor, Vol. 4

Pascal Procedures

Keyboard Wars!

Steven Sheets
Contributing Editor
Hoffman Estates, IL
MacTutor Vol. 4 No. 1

ADB, Keyboards & TypingBattle

One of the most obvious change to the Mac SE and the Mac // has been the redesign of the keyboard and mouse interface. There are no longer 2 separate interfaces for the mouse and the keyboard, each of which could handle only 1 device. Instead the mouse and keyboards are connected through the new Apple Desktop Bus (or ADB). All of Apple's new computers (Mac SE, Mac //, Apple //gs) use ADB. This interface allows multiple ADB devices to be connected to the computer at the same time. Thus a Mac SE could have 2 keyboards, a mouse and a third-party ADB device (perhaps a graphics tablet) all connected to the computer at the same time. While most programs will recognize a key being pressed no matter what keyboard (an A is always an A), a program can be written that recognizes the difference between keyboards and acts accordingly. This article will expand on various ideas involving the ADB, the new keyboards, and the sample program. The sample program that is provided is TypingBattle, a multiple keyboard typing contest.

Apple Desktop Bus

The Apple Desktop Bus is a low-speed input-only interface for the new computers from Apple. The Mac SE and the //gs have a single ADB port while the Mac // has two ports. Each ADB device (except for the Mouse which must be the last item on a ADB line) is a pass through device, similar to the SCSI interface. The first ADB keyboard is plugged into the computer's ADB port, the next ADB keyboard is plugged into the previous keyboard (or a second computer port if there is one). Logically up to 16 ADB devices can be connected to a single Computer. In practice, however, the signal is not powerful enough to go through that many devices. Depending on what style of keyboard is being used, currently 3-5 keyboards and a mouse is the maximum configuration.

ADB devices communicate with the computer using the ADB Manager and special ADB commands. Normally ADB communication is the domain of the ADB device drivers. The Macintosh Start Manager finds all the ADB devices connected to the computer and places that info into an ADB device table. It then initializes any ADB device drivers in the System file. These drivers (stored as 'ADBS' resources) will then handle any ADB communication.

The Mac // and Mac SE System Files contain two standard ADB device drivers, the mouse driver and the universal keyboard driver. The mouse driver handles all mouse movements. It also handles all clicking of the mouse button and issues standard MouseDown and MouseUp events to the Event manager. While two mice can be connected to the Mac //, the mouse driver was not designed to be able to differentiate between mice or pass this information over to the application. No information is passed in the MouseUp or MouseDown event record to explain which Mouse was clicked. Still, for a bit of amusement, connect to two mice to the Mac // and give them to two different people. The cursor will move with each move of either mouse, causing much frustration.

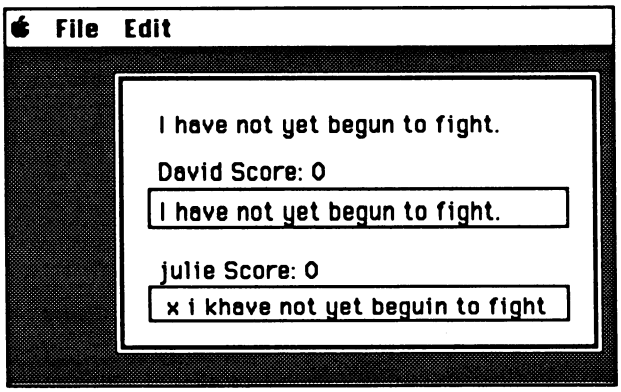


Fig. 1 Two keyboards battle for typing supremacy

The ADB universal keyboard driver handles any keys being pressed or released on an ADB keyboard. It generates a KeyDown or a KeyUp event for the Event Manager. However unlike the mouse driver, the keyboard driver passes along information explaining, not only what key was pressed, but on which keyboard. Prior to the ADB, the Message field of a KeyDown/KeyUp Event record contained the ASCII Character in the first byte and the Virtual Key Code in the second byte. With ADB, the Message field's first and second bytes are the same, while the third bye contains the ADB address of the keyboard that was used. This information is the basis of the sample program, TypingBattle. Two global byte variables are now associated with the ADB Manager; KbdLast, the ADB address of the last keyboard used and KbdType, the Keyboard type of the last keyboard used.

While the sample program only uses the Event record for information about the multiple keyboards, the ADB manager can be accessed directly. The CountADB function returns the total number of ADB devices connected to the computer. GetIndADB (based on the device count from 1 to CountADB number) and GetADBInfo (based on the ADB address of a device) return information about the Device Type, ADB address, service routine address and data area address. Other ADB Manager calls (ADBReInit, ADBOp and SetADBInfo) should not be used directly by an application. Any non-keyboard or non-mouse type of ADB interface would use these commands in it's own ADB device driver.

© The Definitive MacTutor, Vol. 4

The usage of the ADB has yet to be fully recognized. For a hardware developer, designing an ADB device has several advantages. For example, the majority of the current graphics tablets available for the Mac connect to the serial ports, requiring special software drivers (which do not always work with all software packages) and the loss of that serial port for other uses. An ADB graphic tablet that emulates an ADB mouse would work without any special software drivers or loss of port. For the same reasons, an ADB Piano Keyboard would easily interface into all existing Music Packages.

Keyboards

There are currently 3 ADB keyboards that can be used on the Mac SE & Mac //. The standard ADB Keyboard is essentially a duplicate of the older non-ADB MacPlus keyboard. It has a standard keyboard layout with cursor keys and a Numeric Keypad. Programs that were written to use the MacPlus cursor keys will function identically on the standard ADB Keyboard.

Most people do not realize the second ADB keyboard will work on the Mac SE & Mac //. The Apple //gs uses the ADB interface. It uses the same Mouse as the Mac SE & Mac //. However the //gs has it's own ADB keyboard. This keyboard is similar in layout to the standard ADB keyboard, except slightly smaller. Remember that any ADB computer can work with any ADB device. While only a large lab would have both the Macintosh and //gs, it is sometime convenient to know that in a rush, either computer can use the other keyboard.

The third ADB keyboard is the Extended ADB Keyboard (sometimes still referred to by it's development code name "Saratoga"). It has all the features of the standard ADB Keyboard plus additional keys. These additional keys include 15 Function keys, a Control Key, and 6 Special purpose keys.

The 15 Function key, called F1 through F15, are across the top of the keyboard. The first four function keys, F1, F2, F3 and F4, are designed to be assigned the functions Undo, Cut, Copy and Paste. These functions are printed next to each key on the keyboard. The Virtual Key Codes (ie. the third Byte of the Longint passed in the Event field of the event record for keyboard events) of these keys are $7A, $78, $63 and $76. New programs should recognize these Virtual Key Codes, and implement the functions accordingly. According to Apple's Guidelines, the F5 thought F15 keys are intended to be defined by the user, not by the Application. Utilities like QuickKeys by CE Software or Apple's promised MacroMaker allow the user to redefine any key to a specific action. Still, if the developer wishes to do so, a program could use these keys for it's own purpose.

The Control key was added for compatibility with applications that communicate with another operating systems (ie. terminal packages). Pressing the Control key sets bit 12 of the modifier field of the event record for keyboard events. Since the Macintosh OS does not use the concept of Control characters, except for a communication application, this key should not be used by Macintosh applications.

The other 6 Special Keys are intended to give the user greater control of his application. Each application must handle each special key in it's own way. The following are the Keys, their Virtual Key Codes and general guidelines for usage.

The Home key ($73) moves the user to the home position in the application. In a word processor, this key may move the scroll bar to the top of the document. In a database, this may move the display to the first field or the first record. The End key ($77) moves the user to the end position in the application. In a word processor, this key may move the scroll bar to the bottom of the document. The Page Up key ($74) and Page Down Key ($79) moves the user to the next page (up or down) in the application. In a word processor, these keys would be equivalent to clicking in the page up (or down) region of the vertical scroll bar. In a database, these keys may move the user to the previous or next record. According to Apple Guidelines, the Home, End, Page Up and Page Down keys should not effect the actual position of the current insertion point.

The Help key ($72) allows the user to request help from the program. A good usage of this key would be invoke the About menu (or Help menu, if one exists). The Forward Delete, or Fwd Del, key ($75) performs similar to the Backspace key. In a word processor, when the Backspace key is pressed, a character before the current insertion point is deleted. Using the Forward Delete key in the same situation, a character after the current insertion point should be deleted. Use this key carefully. It is a bad idea to allow the user to delete something he can not see (for example, in a database, delete the next unseen record).

TypingBattle

TypingBattle is a sample program designed to show how to handle more than one keyboard. It also implements the special keys of the Apple Extended Keyboard. The game is a simple typing speed contest. First the program prompts each user to type his name from his keyboard. This tells the application the number of players, the name of each player and the ADB address of each player's keyboard. To start a round, each player must press the return key (so that no one is surprised). Roughly three seconds later, a random typing Sample is shown. The first player to correctly type it (including correct punctuation and case), and then press return, wins the round. The player can backspace over any mistakes, up until the time he presses return. Then, his line is judged for correctness. The round continues until a winner finishes or everyone has entered a mistake. The users can use the mouse to select a new round or start a new configuration. The About Menu or the Help Key will display the About Box.

Last Comments

- The program uses MultiLine, a simple Edit unit. Text Edit was not designed to have multiple edit fields open at the same time.
- Modifying the Resource source file or ResEdit can be used to add additional typing Samples.
- Whenever connecting new ADB devices, always power down the computer. A simple reset is not enough for the system to recognizes when new ADB device have been connected or detached.
- In the food for thought area, the sample program show how to use multiple keyboards with a single

screen. What about multiple keyboards and multiple screens? The ADB working together with the Mac / / multiple Video cards would give a true multiuser interface to the Mac. For fun, a MazeWar's type of game would be simple to do. On the serious side, a database package that handled multiple screens/keyboards would have it's advantages. The hardware would be much cheaper than setting up multiple Mac's over Appletalk, while the performance might even increase (no time spent communicating between computers). Any takers?

```
{  TypingBattle by Steve Sheets 11/15/87}
{  Simple Demonstration of Multible Keyboards. }
PROGRAM TypingBattle;
   USES
      MultiLine;    {Unit to handle simple editing}
   CONST
      MaxAllow = 5;     {Max Number of Players}
      AboutID = 600;    {Various Resource IDs}
      TextID = 600;
      LineID = 601;
      AppleMenuID = 1;
      FileMenuID = 2;
      EditMenuID = 3;
      WindowH = 480;    {Placement Constants}
      VEdge = 40;
      CenterTop = 40;
      Edge = 10;
      Hi = 20;
      PlayOff = 50;
      TTop = -15;
      RTop = 5;
      WTop = 35;
      TimeCount = 180; {Timer to start round}
{Variables: Menus, Done Flag, Number Players, Main Window,
Status of game, Number of Samples, Number Players done, lots
strings, arrays holding Players Scores, Who's Done, Names &
Bus IDs and finally Edit fields holding Message, Names &
Text.}
   VAR
      AppleMenu, FileMenu, EditMenu : MenuHandle;
      Done : boolean;
      theNum : integer;
      MyWindow : WindowPtr;
      Status, NumSamples, NumDone : integer;
      WelStr, ScoreStr, PreStr, WinStr, PressStr : str255;
      NoOneStr, EnterStr, SepStr, theTitle : str255;
      Score : ARRAY[1..MaxAllow] OF integer;
      isDone : ARRAY[1..MaxAllow] OF boolean;
      theName : ARRAY[1..MaxAllow] OF str255;
      theBus : ARRAY[1..MaxAllow] OF integer;
      MessRec : MLRec;
      NameRec, TextRec : ARRAY[1..MaxAllow] OF MLRec;

{Returns Number Of Samples (ie. Number of strings in the STR#
resource.}
   FUNCTION GetNumSamples : integer;
      TYPE
         WP = ^integer;
         WH = ^WP;
      VAR
         W : WH;
   BEGIN
      W := POINTER(GetResource('STR#', TextID));
      GetNumSamples := W^^;
      ReleaseResource(POINTER(W));
   END;

{Given M (Longint Message field of Keydown event), return B
(Bus ID), V (Virtual code) & C (Key pressed).}
   PROCEDURE CalcKey (M : longint;
                      VAR B, V : integer;
                      VAR C : char);
   BEGIN
      C := Chr(M MOD 256);
      B := (M DIV 65536) MOD 256;
      V := (M DIV 256) MOD 256;
   END;

{Do About Box.}
   PROCEDURE DoAbout;
      VAR
         n : integer;
   BEGIN
      n := Alert(AboutID, NIL);
   END;

{Initialize variables}
   PROCEDURE DoSetup;
   BEGIN
      GetIndString(WelStr, LineID, 1);
      GetIndString(ScoreStr, LineID, 2);
      GetIndString(PreStr, LineID, 3);
      GetIndString(WinStr, LineID, 4);
      GetIndString(PressStr, LineID, 5);
      GetIndString(NoOneStr, LineID, 6);
      GetIndString(theTitle, LineID, 7);
      GetIndString(EnterStr, LineID, 8);
      GetIndString(SepStr, LineID, 9);
      NumSamples := GetNumSamples;
      AppleMenu := GetMenu(AppleMenuID);
      AddResMenu(AppleMenu, 'DRVR');
      InsertMenu(AppleMenu, 0);
      FileMenu := GetMenu(FileMenuID);
      InsertMenu(FileMenu, 0);
      EditMenu := GetMenu(EditMenuID);
      InsertMenu(EditMenu, 0);
      DrawMenuBar;
      InitCursor;
      MyWindow := NIL;
      Done := false;
   END;

{Given an integer H & V, make a centered rectange R.}
   PROCEDURE MakeRect (VAR R : rect;
                       h, v : integer);
      VAR
         N : integer;
   BEGIN
      N := (screenbits.bounds.right - screenbits.bounds.left -
H) DIV 2;
      SetRect(R, N, VEdge, N + H, VEdge + V);
   END;

{Handle Special Keys, return true if none was pressed.  In
this case, only handle Help by calling About Box.  Note the
Hex Codes.}

   FUNCTION NotSpecKeys (Virtual : integer) : boolean;
   BEGIN
      IF Virtual = $72 THEN
         BEGIN
            DoAbout;
            NotSpecKeys := false;
         END
      ELSE
         NotSpecKeys := true;
   END;

{Finds out who is playing (ie. set Names & Bus IDs) or quit
game (ie. Done true).}

   PROCEDURE GetPlayers;
```


© The Definitive MacTutor, Vol. 4

```
CONST
   ConV = 85;
   ConH = 360;
   DLOff = 15;
   kOff = 25;
   kTop = 15;
   kHi = 20;
   LineLeft = 20;
   butBot = -15;
   OKLeft = 90;
   QUITLeft = 210;
   Voff = 15;
VAR
   tempPort : Grafptr;
   myW : WindowPtr;
   cont, F2 : boolean;
   tempEvent : EventRecord;
   tempWindow : windowptr;
   tempCode, tempVirtual : integer;
   tempChar : char;
   OKcon, QUITcon, tempCon : ControlHandle;
   Lines : ARRAY[1..MaxAllow] OF MLRec;

PROCEDURE DoBox (V : integer);
   VAR
      R2 : Rect;
BEGIN
   WITH Lines[V] DO
      BEGIN
         R2.left := Fr.left - 1;
         R2.right := Fr.right + 1;
         R2.top := Fr.top - 1;
         R2.bottom := Fr.bottom + 1;
         IF V > theNum THEN
            PenPat(Gray);
         FrameRect(R2);
         PenPat(Black);
      END;
END;

PROCEDURE CheckOK;
BEGIN
   IF (theNum > 0) AND (theNum <= MaxAllow) THEN
      HiliteControl(OKcon, 0)
   ELSE
      HiliteControl(OKcon, 255);
END;

PROCEDURE RegKey (V : integer;
                  C : Char);
   VAR
      count, L : integer;
BEGIN
   count := 0;
   FOR L := 1 TO theNum DO
      IF theBus[L] = v THEN
         count := L;
   IF count = 0 THEN
      BEGIN
      IF (theNum < MaxAllow) AND (Ord(C) >= 32) THEN
         BEGIN
            theNum := theNum + 1;
            theBus[theNum] := V;
            Lines[theNum].St := ' ';
            Lines[theNum].St[1] := C;
            Lines[theNum].Cr := '_';
            MLreset(Lines[theNum]);
            DoBox(theNum);
            CheckOk;
         END;
      END
   ELSE
      BEGIN
      MLchar(Lines[count], C);
      IF Lines[count].St = '' THEN
         BEGIN
         IF count <> theNum THEN
            BEGIN
            FOR L := count TO theNum - 1 DO
               BEGIN
                  MLtext(Lines[L], Lines[L + 1].St);
                  theBus[L] := theBus[L + 1];
               END;
            Lines[theNum].St := '';
            END;
         Lines[theNum].Cr := ' ';
         MLreset(Lines[theNum]);
         theNum := theNum - 1;
         DoBox(theNum + 1);
         CheckOk;
         END
      END;
   END;

   PROCEDURE DoGPSetup;
      VAR
         nn, count : integer;
         S : str255;
         Bx : rect;
BEGIN
   nn := ConV + (MaxAllow * kOff);
   MakeRect(Bx, ConH, nn);
   myW := NewWindow(NIL, Bx, '', true, 1, POINTER(-1),
false, 0);
   SetPort(myW);
   SetRect(Bx, OKLeft, nn - 20 + butBot, OKLeft + 60, nn +
butBot);
   GetIndString(S, LineID, 10);
   OKcon := NewControl(myW, Bx, S, true, 0, 0, 0, 0, 0);
   SetRect(Bx, QUITLeft, nn - 20 + butBot, QUITLeft + 60,
nn + butBot);
   GetIndString(S, LineID, 11);
   QUITcon := NewControl(myW, Bx, S, true, 0, 0, 0, 0, 0);
   cont := false;
   theNum := 0;

   FOR count := 1 TO MaxAllow DO
   BEGIN
   nn := (count * kOff) + DLOff;
   SetRect(Bx, LineLeft, nn, ConH - LineLeft, nn + kHi);
      MLinit(Lines[count], '', '', ' ', Bx, false);
   END;
END;

BEGIN
DoGPSetup;
CheckOk;
REPEAT
SystemTask;
IF GetNextEvent(everyEvent, tempEvent) THEN
BEGIN
   IF tempEvent.what = mouseDown THEN
   BEGIN
   F2 := true;
   IF FindWindow(tempEvent.where, tempWindow) = inContent
THEN
   BEGIN
      GlobalToLocal(tempEvent.where);
      IF FindControl(tempEvent.where, myW, tempCon) <> 0 THEN
         IF TrackControl(tempCon, tempEvent.where, NIL) <> 0
THEN
            BEGIN
               IF tempCon = QUITcon THEN
                  BEGIN
                     Done := true;
                     F2 := false;
                  END
               ELSE IF tempCon = OKcon THEN
```

```
                BEGIN
                   Cont := true;
                   FOR tempCode := 1 TO theNum DO
                      theName[tempCode] :=
Lines[tempCode].St;
                   F2 := false;
                END;
             END
          ELSE
             F2 := false;
   END;
   IF F2 THEN
      sysbeep(1);
   END;
   IF tempEvent.what = keydown THEN
      BEGIN
      CalcKey(tempEvent.message, tempCode, tempVirtual,
tempChar);
      IF NotSpecKeys(tempVirtual) THEN
         RegKey(tempCode, tempChar);
      END;
   IF tempEvent.what = updateEvt THEN
      IF myW = WindowPtr(tempEvent.message) THEN
         BEGIN
            GetPort(tempPort);
            SetPort(myW);
            BeginUpdate(myW);
            MoveTo((ConH - StringWidth(EnterStr)) DIV 2,
kTop + Voff);
            DrawString(EnterStr);
            FOR tempCode := 1 TO MaxAllow DO
               BEGIN
                  MLupdate(Lines[tempCode]);
                  DoBox(tempCode);
               END;
            DrawControls(myW);
            EndUpdate(myW);
            SetPort(tempPort);
         END;
END;
UNTIL cont OR done;
KillControls(myW);
DisposeWindow(myW);
END;

{Dispose Window (if any), get Players (if any), if so create
the Window and}
{  Edit fields for the next game.}
   PROCEDURE DoConfigure;
      VAR
         tempRect : Rect;
         flag : boolean;
         count : integer;
         Bx : rect;
   BEGIN
      IF MyWindow <> NIL THEN
         BEGIN
            DisposeWindow(MyWindow);
            MyWindow := NIL;
         END;

      GetPlayers;

      IF NOT done THEN
         BEGIN
            MakeRect(tempRect, WindowH, WTop + (theNum *
PlayOff));
            MyWindow := NewWindow(NIL, tempRect, theTitle,
false, 1, POINTER(-1), false, 0);
            SetPort(MyWindow);
            Bx.left := Edge;
            Bx.right := WindowH - Edge;
            Bx.top := Edge;
            Bx.bottom := Bx.top + Hi;
            MLinit(MessRec, '', '', ' ', Bx, false);
            FOR count := 1 TO theNum DO
               BEGIN
                  Bx.top := (count * PlayOff) + TTop;
                  Bx.bottom := Bx.top + Hi;
                  Score[count] := 0;
                  MLinit(NameRec[count], CONCAT(theName[count],
ScoreStr), '', ' ', Bx, false);
                  Bx.top := (count * PlayOff) + RTop;
                  Bx.bottom := Bx.top + Hi;
                  MLinit(TextRec[count], '', '', ' ', Bx, true);
                  END;
            ShowWindow(MyWindow);
         END;
   END;

{Start a game by setting status to 1, setting correct message,
clearing score,}
{  text and done flags.}
   PROCEDURE DoStart;
      VAR
         count : integer;
         tempPort : GrafPtr;
   BEGIN
      GetPort(tempPort);
      SetPort(MyWindow);
      Status := 1;
      MLtext(MessRec, WelStr);
      FOR count := 1 TO theNum DO
         BEGIN
            Score[count] := 0;
            MLtext(NameRec[count], '0');
            MLtext(TextRec[count], '');
            IsDone[count] := false;
         END;
      SetPort(tempPort);
   END;

{Handle updating the window by calling Edit fields update.}
   PROCEDURE DoUp;
      VAR
         count : integer;
         tempR : rect;
   BEGIN
      SetRect(tempR, 0, 0, 1000, 1000);
      EraseRect(tempR);
      MLupdate(MessRec);
      FOR count := 1 TO theNum DO
         BEGIN
            MLupdate(NameRec[count]);
            MLupdate(TextRec[count]);
         END;
   END;

{Flush the Event buffer of Keydowns.}
   PROCEDURE FlushKeys;
   BEGIN
      FlushEvents(keyDownMask, 0);
   END;

{Depending on Game Status, handle the Key down.  Who is
Players number (not bus ID).  Status 1 is everyone waiting for
all Players to press return for the next round.Status 2 has
everyone playing.}
PROCEDURE DoKey (Who : integer;
                 Key : char);
VAR
      ll : longint;
      Sp : str255;
      count : integer;
      dummy : boolean;
      tempPort : GrafPtr;
BEGIN
   GetPort(tempPort);
```

```
   SetPort(MyWindow);
   SetPort(MyWindow);
   IF (Status = 1) AND (Key = Chr(13)) THEN
   BEGIN     (Player Pressed return.)
      IsDone[who] := true;
      dummy := true;
      FOR count := 1 TO theNum Do
         dummy := dummy AND IsDone[count];

(If everyone pressed return, wait awhile, then display the
Sample, flush any old key events & go to Status 2.)

      IF dummy THEN
      BEGIN
         MLtext(MessRec, PreStr);
         ll := Tickcount;
         NumDone := 0;
         FOR count := 1 TO theNum DO
            BEGIN
               IsDone[count] := false;
               MLtext(TextRec[count], '');
            END;
         count := (Random MOD NumSamples) + 1;
         GetIndString(Sp, TextID, count);
         WHILE tickcount < ll + TimeCount DO
                  ;
         MLtext(MessRec, Sp);
         Status := 2;
         FlushKeys;
      END;
   END
   ELSE IF Status = 2 THEN
      BEGIN
      (If player has not pressed return,)
      IF NOT IsDone[who] THEN
         BEGIN
            MLchar(TextRec[who], Key);
            (Handle the Key.)
            IF Key = Chr(13) THEN
               BEGIN
               (If return, he is done.)
                  NumDone := NumDone + 1;
                  IsDone[who] := true;
                  IF EqualString(TextRec[who].St,
MessRec.St, true, false) THEN
                  BEGIN
                  (Handle him winning.)
                  Status := 1;
                  Score[who] := Score[who] +
LENGTH(MessRec.St);
                  MLtext(MessRec, CONCAT(WinStr,
theName[who], PressStr));
                  NumToString(Score[who], Sp);
                  MLtext(NameRec[who], Sp);
                  FOR count := 1 TO theNum DO
                     IsDone[count] := false;
                  END
                  ELSE IF NumDone = theNum THEN
                  BEGIN (Handle no one winning.)
                     Status := 1;
                     MLtext(MessRec, NoOneStr);
                     FOR count := 1 TO theNum DO
                        IsDone[count] := false;
                  END;
            END;
      END;
   END;
SetPort(tempPort);
END;

(Handle Menu.)
PROCEDURE MainMenu (tempResult : LONGINT);
   VAR
         tempInteger : integer;
         tempStr : str255;
BEGIN
   tempInteger := LoWord(tempResult);
   CASE HiWord(tempResult) OF
   AppleMenuID :
      IF tempInteger = 1 THEN
         DoAbout
      ELSE
         BEGIN
            GetItem(appleMenu, tempInteger, tempStr);
            tempInteger := OpenDeskAcc(tempStr);
         END;
   FileMenuID :
      IF tempInteger IN [1, 2] THEN
         BEGIN
            IF tempInteger = 2 THEN
               DoConfigure;
            IF NOT done THEN
               DoStart;
         END
      ELSE IF tempInteger = 4 THEN
         Done := true;
   EditMenuID :
      IF NOT SystemEdit(tempInteger - 1) THEN
         sysbeep(1);
   OTHERWISE
   END;
   HiliteMenu(0);
END;

(Main Event Loop.)
PROCEDURE DoMainLoop;
   VAR
         tempEvent : EventRecord;
         tempWindow : windowptr;
         tempCode, tempBus, tempVirtual : integer;
         tempPort : Grafptr;
         tempChar : Char;
         tempRect : rect;
BEGIN
REPEAT
SystemTask;
IF GetNextEvent(everyEvent, tempEvent) THEN
BEGIN
CASE tempEvent.what OF
mouseDown :
   BEGIN
      tempCode := FindWindow(tempEvent.where, tempWindow);
      CASE tempCode OF
      inMenuBar :
         MainMenu(MenuSelect(tempEvent.where));


      inSysWindow :
         SystemClick(tempEvent, tempWindow);
      inContent :
         IF tempWindow <> FrontWindow THEN
            SelectWindow(tempWindow);
      inDrag :
         IF (MyWindow = tempWindow) AND (tempWindow <> NIL)
THEN
         BEGIN
         SetRect(tempRect, -32000, -32000, 32000, 32000);
         DragWindow(tempWindow, tempEvent.where, tempRect);
         END;
      OTHERWISE
      END; ( of tempCode case )
   END; ( of mouseDown )
keydown :
   BEGIN
   CalcKey(tempEvent.message, tempBus, tempVirtual, tempChar);
   IF BitAnd(tempEvent.modifiers, cmdKey) <> 0 THEN
      MainMenu(MenuKey(tempChar))
   ELSE IF NotSpecKeys(tempVirtual) THEN
```

```
      FOR tempCode := 1 TO theNum DO
         IF (theBus[tempCode] = tempBus) AND (theNum >
(tempCode - 1)) THEN
            DoKey(tempCode, tempChar)
   END;
updateEvt :
   BEGIN
      tempWindow := WindowPtr(tempEvent.message);
      GetPort(tempPort);
      SetPort(tempWindow);
      BeginUpdate(tempWindow);
      IF (tempWindow = MyWindow) AND (tempWindow <> NIL)
THEN
         DoUp;
      EndUpdate(tempWindow);
      SetPort(tempPort);
   END;
OTHERWISE
END;
END;
UNTIL Done;
END;

{Game over.}
   PROCEDURE DoQuit;
      VAR
         n : integer;
   BEGIN
      DeleteMenu(AppleMenuID);
      DeleteMenu(FileMenuID);
      DeleteMenu(EditMenuID);
      IF MyWindow <> NIL THEN
         DisposeWindow(MyWindow);
   END;

{Main Body}
BEGIN
   DoSetup;
   DoAbout;
   DoConfigure;
   IF NOT done THEN
      BEGIN
         DoStart;
         DoMainLoop;
      END;
   DoQuit;
END.
```

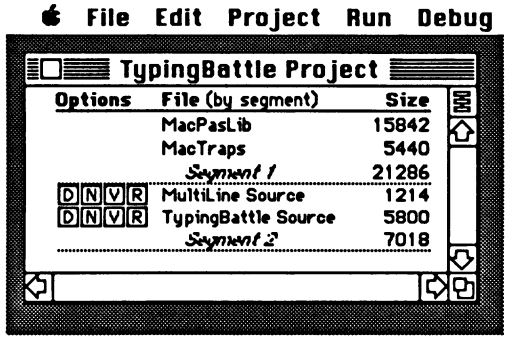
```
{  MultiLine Unit by Steve Sheets 11/15/87  }
{  Simple Edit Fields for Multible Keyboards.  }
UNIT MultiLine;

INTERFACE
{Data Type containing Text Str, Prompt Str, Cursor Char,
Cursor Width, Frame Rect, Frame Flag, Horizontal Start
Position, Horizontal Current Postion, Horizontal Prompt
Position, Vertical Start Postion.}
   TYPE
      MLRec = RECORD
            St, Pr : Str255;
            Cr : char;
            CW : integer;
            Fr : rect;
            FF : boolean;
            StartH, CurH, PrH, StartV : integer;
         END;

{Draws the Rec Edit Field}
   PROCEDURE MLupdate (Rec : MLRec);

{Resets the Rec Edit Field, after someone has changed a
setting.}
   PROCEDURE MLreset (VAR Rec : MLRec);

{Initalize the Rec Edit Field}
   PROCEDURE MLinit (VAR Rec : MLRec;
                              Prompt : str255;
                              Tx : str255;
                              Curs : char;
                              Box : rect;
                              FrFlag : boolean);

{Handle a Key (Printable Keys & Backspace)}
   PROCEDURE MLchar (VAR Rec : MLRec;
                              C : char);

{Reset the Text Str to S (calls MLReset).}
   PROCEDURE MLtext (VAR Rec : MLRec;
                              S : str255);

IMPLEMENTATION

   PROCEDURE MLupdate;
      VAR
         R2 : rect;
   BEGIN
      WITH Rec DO
         BEGIN
            IF FF THEN
               BEGIN
                  FrameRect(Fr);
                  R2.left := Fr.left + 1;
                  R2.right := Fr.right - 1;
                  R2.top := Fr.top + 1;
                  R2.bottom := Fr.bottom - 1;
                  EraseRect(R2);
               END
            ELSE
               EraseRect(Fr);
            MoveTo(StartH, StartV);
            IF Pr <> '' THEN
               DrawString(Pr);
            DrawString(St);
            IF CW > 0 THEN
               DrawChar(Cr);
         END;
   END;

   PROCEDURE MLreset;
   BEGIN
      WITH Rec DO
         BEGIN
            CurH := StartH;
            IF Pr <> '' THEN
               CurH := CurH + StringWidth(Pr);
            PrH := CurH;
            IF St <> '' THEN
               CurH := CurH + StringWidth(St);
            IF Cr <> ' ' THEN
               CW := CharWidth(Cr)
            ELSE
               CW := 0;
         END;
      MLupdate(Rec);
   END;

   PROCEDURE MLinit;
   BEGIN
      WITH Rec DO
         BEGIN
            Fr := Box;
            St := Tx;
            Pr := Prompt;
            StartH := Fr.left + 5;
            StartV := Fr.top + 16;
            FF := FrFlag;
            Cr := Curs;
         END;
```


© The Definitive MacTutor, Vol. 4

```
      MLreset(Rec);
   END;

   PROCEDURE MLchar;
      VAR
         Small : str255;
         L : integer;
   BEGIN
      WITH Rec DO
         IF Ord(C) = 8 THEN
            BEGIN
               L := Length(St);
               IF L > 0 THEN
                  BEGIN
                     CurH := CurH - CharWidth(St[L]);

                     TextMode(srcBic);
                     MoveTo(CurH, StartV);
                     DrawChar(St[L]);
                     IF CW > 0 THEN
                        DrawChar(Cr);

                     TextMode(srcOr);
                     MoveTo(CurH, StartV);
                     IF CW > 0 THEN
                        DrawChar(Cr);
                     Delete(St, L, 1);
                  END;
            END
         ELSE IF (Ord(C) >= 32) THEN
            BEGIN
               IF CharWidth(C) + CW + CurH < Fr.right THEN
                  BEGIN
                     MoveTo(CurH, StartV);
                     IF CW > 0 THEN
                        BEGIN
                           TextMode(srcBic);
                           DrawChar(Cr);
                           TextMode(srcOr);
                           MoveTo(CurH, StartV);
                        END;
                     DrawChar(C);
                     CurH := CurH + CharWidth(C);
                     IF CW > 0 THEN
                        DrawChar(Cr);

                     Small := '@';
                     Small[1] := C;
                     St := CONCAT(St, Small);
                  END;
            END;
   END;

   PROCEDURE MLtext;
   BEGIN
      Rec.St := S;
      MLReset(Rec);
   END;
END.
```

TypingBattle.Rsrc

```
Type MENU
     ,1
\14
     About TypingBattle...
     (-

     ,2
File
     New Game
     New Configuration
     (-
     Quit

     ,3
Edit
     Undo/Z
     (-
     Cut/X
     Copy/C
     Paste/V
     Clear

Type DITL
     ,600
3

button
130 170 150 230
OK

staticText
20 18 40 382
Welcome to TypingBattle by Steve Sheets for MacTutor

staticText
60 18 110 382
TypingBattle is a Multiplaying Typing Contest designed to be
used ++
on a Mac SE or Mac // with 1 or more ADB Keyboards.

Type ALRT
     ,600
40 56 210 456
600
4444

Type STR#
     ,601
11
Everyone press return to start a new sequence.
 Score:
Prepare to type:
The winner is
.  Everyone press return to start a new sequence.
No one has finished.  Everyone press return to start a new
sequence.
TypingBattle
Enter your name from the correct keyboard:
:
OK
Quit

Type STR#
     ,600
5
Now is the time for all good men
ABCDEFGHIJKLMNOPQRSTUVWXYZ
The quick red fox jumped over the lazy brown cow.
I have not yet begun to fight.
An Apple a day keeps the doctor away.
```



The Visiting Developer

Format Drivers Bring Freedom of Revison

David Dunham
James Hopper
Goleta, CA
MacTutor Vol. 1 No. 1

Introduction

Data interchange is a real problem. Everyone likes to save information in their own format, and no one wants to deal with anyone else's format (especially not to write it!). Programs like the original version of *Acta*, David's desk accessory outline processor, support only limited file format interchange, usually via a separate translator program. Such translation programs are very inconvenient, but there are reasons for writing them. First, the extra code required to read additional file formats can significantly enlarge a program while adding support for formats which most users never need. It's better to separate this code from the more commonly-used parts of the program. Also, the intermediate file is necessary because many programs don't use the public clipboard (desk scrap) for formatted text. Second, many of these formats are considered proprietary and are not generally available, for example Microsoft *Word* 1.05. Finally, there are a lot of file formats out there — most of us are simply too lazy to cover all of them.

Many people work with a variety of editors, word processors, and desk accessories, and need the ability to exchange files between them. Separate translation programs such as the one used with *WriteNow* are simply too inconvenient. Macintosh users have grown accustomed to (and deserve) a higher degree of integration between programs. One way this can be done is to 'teach' the program what the data must look like by having the program look-up the format externally to itself. In this way, the program does not have to be revised to read or write a new format whenever someone releases a hot new product. In this article, we will discuss an example of such a technique used in *Acta* 2.0, that can be applied to other developer's software projects to ease the burden of being compatible. *[It is hoped this article will spur someone to write a similar article on how Pagemaker compatibility can be achieved by new word processors without having to wait for Aldus to release a new Pagemaker version. Wouldn't it be neat if the word processor or graphics developer could make his product Pagemaker compatible without Aldus having to do anything? -Ed]*

Acta 2.0 handles additional data formats with **format drivers**, files which, like printer drivers, reside in the system folder. Because they're separate from the main program, users can install only the formats required by dragging these driver files to the system folder. As new word processors come out, the new file formats can be supported with new format drivers, rather than a new version of *Acta*. Given the file format documentation for the new word processing program, drivers to support new formats can be produced by third party programmers such as Jim.

David will first explain format drivers using C as the target language. Jim will then discuss a simple input format driver in Pascal, focusing on the glue required to link functions written in C to units written in Pascal. Although this article deals with format drivers as implemented in *Acta*, the concept of format drivers can be adopted in most programs.

Additional technical information on *Acta* can be found in a short document describing the format of *Acta* files, available from online services such as Delphi or CompuServe, or from Symmetry Corp. For convenience, it is included at the end of this article. This information would be useful if you're implementing your own *Acta* format driver, but you won't need it to read this article.

Input Drivers

Acta supports two kinds of format drivers: input and output. A file can contain resources for both, but we'll start by looking at input drivers. An input driver's job is to read a file, parse the data, and pass it to *Acta* as **topics** (*Acta*'s smallest data object). *Acta* handles everything else, which keeps down the overhead in each format driver. We'll use the input driver a_TEXT as an example format driver. This example was written in C and compiled using *MPW* C. This driver reads text files, splitting the text into topics at return characters.

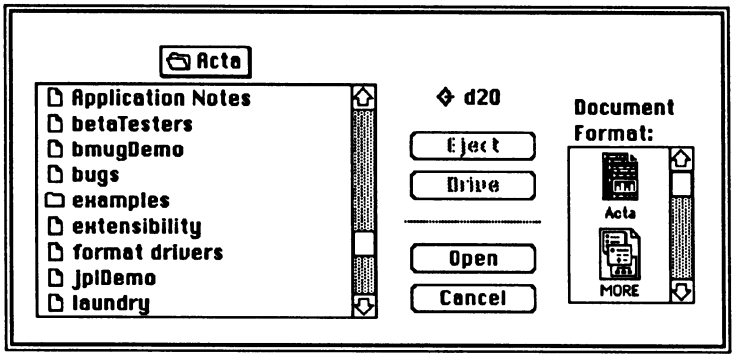
The driver a_TEXT is composed of a variety of resources stored in the resource fork of the file a_TEXT. The file type for a format driver is 'ACTF,' which is how *Acta* recognizes it as a format driver. The file creator 'Atxt' associates a unique icon with the driver file in the Finder using the usual additional (bundle) resources. We discuss other resources in the following paragraphs.

ACTf — The code for the driver is stored as a resource of type ACTf. It's actually a standard code resource which *Acta* loads using GetResource.

ICN# — An input driver file must contain an ICN# 64 resource which, along with the file name, identifies a particular format driver. The user selects an input driver from a scrolling icon list (see Appendix) as part of *Acta*'s **Open...** dialog (see illustration below). To avoid confusion with application names, format driver files start with 'a_' by convention; however, this prefix is not displayed in the scrolling icon list.

FILT — If a resource FILT 64 is present, it's assumed to be a code resource containing a file filter function, defined as in SFGetFile:

© The Definitive MacTutor, Vol. 4




```
pascal short fileFilter(pb) ParmBlkPtr pb;
```

The driver author provides this filter so *Acta* can open multiple file types using the format driver.

TYPE — If there is no FILT 64, *Acta* determines which files to open by examining the input driver's resource TYPE 64, which is four ASCII characters and specifies a single file type to be converted.

Once the user decides to open a file, *Acta* calls the driver, assuming it's a C routine defined as is read_text() in Listing 1.

```
/*
Read TEXT documents into Acta
Written by David Dunham
©1987 Maitreya Design - code may be freely used
in Acta format drivers
*/

typedef int      word;     /* Synonyms for VAX-hackers */
typedef unsigned char  byte;

#define TAB      '\t'
#define CR    '\r'

#define NIL      (0L)
#define TRUE     (-1)
#define FALSE    0

#define TEXTtype 1
#define PICTtype 2

struct BLOCK_ATT {  /* Attribute portion of a topic */
   word   bType;        /* TEXTtype, PICTtype, … */
   word   bAttributes;  /* 0 */
   word   bFont;
   word   bStyle;
   word   bSize;
   byte   bColor;       /*Triangle color: index to COL#
                        resource, 0 = black */
   byte   filler;       /* 0 */
   byte   firstLine;    /* Show first line only? 0xFF or 0 */
   byte   expanded;     /* Subtopics visible? 0xFF or 0 */
   word   hidden;       /* Is block hidden (by Collapse)? count
                        of hide depth */
};
typedef struct BLOCK_ATT   BLOCK_ATT;

/*
READ_TEXT - Read generic TEXT file to Acta outline
      infil     : refNum of input file
      resfil    : refNum of input file's resource fork
      att       : default topic attributes
      add_topic : pointer to routine to call
*/
OSErr read_text(infil,resfil,att,add_topic)
word infil, resfil; BLOCK_ATT *att; ProcPtr add_topic; {
   register char    **text;       /* Handle for  text */
   register OSErr   error;
   long             count;        /* Used in FS calls */
   register word    level;        /* Topic's level */
   word   i_count, last_count, spaces;
   word   last_level;       /* Level of previous topic */
   word   space_indent;     /* No. of spaces = to a tab */
   char   c;
   long   size;             /* Length of text */
   Handle h;
   word   oldResFile;

   oldResFile = CurResFile(); /* Previous top resource
file */
   UseResFile(resfil);        /* Just look in document */
   h = GetIndResource('EFNT',1);
                              /* MDS Edit/QUED/miniWRITER
font */
   if (h != NIL) {            /* There is such a resource */
      att->bSize = **(word **)h;
      GetFNum(*(char **)h + 2,&att->bFont);
   }
   UseResFile(oldResFile);    /* Restore search order */
   level = -1;          /* We've never parsed a topic */
last_level = 0;
space_indent = 5;/* SHOULD be rsrc; 5 for MORE 1.0 */
last_count = i_count = spaces = 0;
text = NIL;
att->bType = TEXTtype;  /* Everything is text! */
while (TRUE) {          /* Keep going until EOF */
                        /* Will this die with empty file? */
   count = 1L;      /* Read a single character */
   error = FSRead(infil,&count,&c);
   if (error) {     /* Probably hit EOF before RETURN */
      if (error == eofErr) return(noErr);
      /* Finished with the file */
      return(error);            /* Uh-oh… */
   }
   if (c == TAB) {
      i_count++;                /* Increment indent count */
      spaces = 0;               /* Start count over */
   } else if (c == ' ') {
      spaces++;
      if (spaces == space_indent) { /* A pseudo-tab */
         i_count++;       /* Increment indent count */
         spaces = 0;          /* Start count over */
      }
   } else {                     /* Not space or tab */
      if (level == -1)          /* Very first topic */
         level = 1;
       else
         level = level + i_count - last_count;
      if (level < 1)
         level = 1;       /* Minimum indent */
      if (level > last_level + 1)
         level = last_level + 1;  /*go one level deeper */
      if (level == 1) {       /* Left-margin topic? */
         att->expanded = TRUE;
         att->hidden = 0;
       } else {           /* Subtopics aren't expanded */
         att->expanded = FALSE;
         att->hidden = level - 2;
                        /* # of collapsed levels above */
      }
      text = NIL;               /* No text so far */
      /* Read until end of line or end of file */
      while (c != CR && error == noErr) {
         if (text == NIL) {     /* First character? */
            text = NewHandle(1L);
            **text = c;         /* Copy it in */
         } else
            PtrAndHand(&c,text,1L); /* Add to the end */
```

```
            count = 1L;          /* Read a single character */
            error = FSRead(infil,&count,&c);
         }
         if (error != noErr && error != eofErr)
            return(error);          /* Uh-oh… */
         if (text == NIL) {         /* Empty topic */
            (*add_topic)(NIL,0L,level,att,NIL);
          } else {
            size = GetHandleSize(text);
            HLock(text);         /* Keep dereference valid */
            (*add_topic)(*text,size,level,att,NIL);
            DisposHandle(text);     /* Free the memory */
            text = NIL;             /* Mark it as empty */
         }
         if (error == eofErr) {
            return(noErr);       /* Finished with the file */
         }
         last_count = i_count;   /* input file's indent */
         i_count = spaces = 0;
         last_level = level;     /* Remember topic's level */
      } /* end if */
   } /* end while */
}
```

«listing 1»

The actual code in our example is straightforward. First, it looks in the resource fork for the EFNT resource (where *MDS Edit*, *QUED*, and *miniWRITER* save a text file's font and size). Then it reads the data fork a character at a time, appending them to the end of a block of memory until it reaches either the end of the file or a return character. When this happens, it calls **add_topic**, a routine in one of *Acta*'s code resources. If there are any errors, it just returns the error to *Acta*, which takes care of informing the user. A driver can display its own error messages, in which case it returns -2048.

Add_topic inserts the data into *Acta*'s family tree structure. It's declared as:

```
/*
ADD_TOPIC - Create a new topic and link it in
      data          : topic data
      len        : length of data
      level         : level of topic (1 = left margin)
      attributes: topic's attributes
      styleHandle   : RESERVED
*/
void add_topic(data,length,level,attributes,styleHandle)
Ptr data; long length; word level; BLOCK *attributes; Handle
styleHandle;
```

The code is compiled into the resource ACTf 64:

```
c a_TEXT.c
link -rt ACTf=64 -sn Main=a_TEXT -c Atxt -t ACTF ∂
   a_TEXT.c.o -o d20:MPW:a_TEXT ∂
   "(Libraries)"Interface.o ∂
   "(CLibraries)"CInterface.o
```

Listing 2 summarizes the resources which make up an input driver, in REZ format:

```
resource 'BNDL' (128) {
   'Atxt',
   0,
   {  /* array TypeArray: 2 elements */
      'ICN#',
      {  /* array IDArray: 1 elements */
         0, 128
      };
      'FREF',
      {  /* array IDArray: 1 elements */
         0, 128
      }
   }
};

resource 'FREF' (128, "a_TEXT") {
   'ACTF',
   0,
   ""
};

data 'Atxt' (0, purgeable) {
   "©1987 Maitreya Design 23 January 1987"
};

data 'TYPE' (64, purgeable) {
   'TEXT'
};
```

and the ICN#

«listing 2 — *Rez* input»

Output Drivers

An output driver is responsible for taking the data from an *Acta* topic and writing it to a file. Again, *Acta* handles most of the overhead, but the process is slightly more complicated, because most word processor formats involve tables and pointers. *Acta* calls the output driver before passing any topic data, once for each topic, and again after all the data. The driver generally builds any tables during the first and last call. For example, a word processor format may begin with a list of paragraphs (conveniently written during the first call) and end with a pointer to free space (best written after all the topics have been output).

The example output driver is a_RTF, which converts an outline to Rich Text Format, Microsoft's interchange format. It creates a file which is as true as possible to the screen appearance of the *Acta* outline. (The RTF specification is available from Microsoft. It's beyond the scope of this article to explain in detail, but it's a format for representing formatting information using only the standard 7-bit printing ASCII characters. Backslashes precede formatting words, and braces group items. This format is also used for Microsoft's MSDOS word processor.)

ACTf — The actual code for the output driver is in the ACTf 0 resource and is loaded by *Acta* using GetResource.

ICN# — An output driver file must contain an ICN# 0 resource which, along with the file name, identifies the format driver to the user. The user selects an output driver from a scrolling icon list (see Appendix) as part of *Acta*'s Save as... dialog (see illustration below). When the user clicks on this icon, *Acta* highlights the ruler and label buttons depending on the RULR 0 resource.

RULR — This is a word, in which the following bits have meaning:

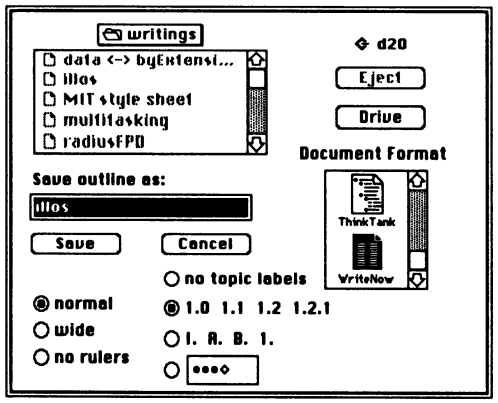
0x8000	format has rulers
0x4000	format doesn't use labels

As usual, don't set unused bits.

TYPE and CREA — The resources TYPE 0 and CREA 0 are OSTypes, 4 ASCII characters *Acta* uses to set the output file's type and creator.

Acta passes a pointer to a parameter block which indicates the topic's level, data, attributes, label, and other information; this is defined in Listing 3.

When the example driver gets the -1 message (indicating the



initial call), it writes the RTF header, including the name of every installed font. When it gets the 0 message (indicating a topic's data), it outputs the topic's style, ruler, label, and data. If it encounters an error, it must return the error (or -2048 if it doesn't want *Acta* to put up an error dialog). Since a code resource can't have globals, it sets already_error in the parameter block. A_RTF ignores the +1 message, which follows all the 0 messages.

```
/*
Outputs Acta outline as Rich Text Format (Word 3.x) document
Written by David Dunham
©1987 Maitreya Design - code may be freely used
in Acta format drivers
*/

typedef int   word;     /* Synonyms for VAX-hackers */
typedef unsigned char  byte;

#define TAB     '\t'
#define CR    '\r'

#define NIL     (0L)
#define TRUE    (-1)
#define FALSE   0

/* Label types */
#define NONE     10
#define DECIMAL  11
#define ROMAN    12
#define BULLETS  14

/* Topic types */
#define TEXTtype 1
#define PICTtype 2

struct BLOCK_ATT {      /* Attribute portion of a topic */
   word   bType;        /* TEXTtype, PICTtype, … */
   word   bAttributes;  /* 0 */
   word   bFont;
   word   bStyle;
   word   bSize;
   byte   bColor;       /* Triangle color: index to COL#
                           resource, 0 = black */
   byte   filler;       /* 0 */
   byte   firstLine;    /* Show first line only? 0xFF or 0 */
   byte   expanded;     /* Subtopics visible? 0xFF or 0 */
   word   hidden;       /* Is block hidden (by Collapse)? count
                           of hide depth */
};
typedef struct BLOCK_ATT   BLOCK_ATT;

struct OUT_PARMS {
   word         fRef;          /* Data fork refNum */
   word         rRef;          /* Resource fork refNum */
   word         message;
   word         level;
   word         n;             /* Topic's absolute number */
   char         *data;
   long         len;           /* Length of data */
   BLOCK_ATT    *att;
   THPrint      print_rec;
   word         already_error;
   char         *label;
   word         labelType;
   word         *raw_label;
   word         rulers;
   word         error_reporting; /* not used */
   Handle       reserved;        /* Currently NIL */
   word         last_level;
   word         private;
};
typedef struct OUT_PARMS   *P_PTR;

/* Function prototypes */
void      file_error();
void      warp();
void      w_byte();
void      w_str();
OSErr     write_RTF();

/*
WRITE_RTF - Write Rich Text Format document
*/
OSErr write_RTF(p) register P_PTR p; {
   char                 *s;
   long                 count;
   OSErr                error;
   Handle               handle;
   register             word   i;
   word                 id, last_space, rsrc_fonts;
   Str255               name, string;
   WindowPtr            front;

   switch (p->message) {
     case -1:
       w_str(p,"\P{\\rtf\\mac");
       /* Output font table (every installed font) */
       w_str(p,"\P{\\fonttbl");
       rsrc_fonts = CountResources('FONT');
       for (i = 1; i <= rsrc_fonts; i++) {
          SetResLoad(FALSE);  /* Don't actually load fonts! */
          handle = GetIndResource('FONT',i);
          GetResInfo(handle,&id,&count,&name);
          SetResLoad(TRUE);       /* Back to normal */
          if (name.length != 0) {
             w_str(p,"\P{\\f");
             NumToString((long)id >> 7,&string);
             w_str(p,&string);
             w_byte(p,' ');
             w_str(p,&name);
             w_byte(p,';');
          }
       }
       w_str(p,"\P}\r");
       break;
     case 0:
```

```
      w_str(p,"\P{\\s10\\f");
      NumToString((long)p->att->bFont,&string);
      w_str(p,&string);
      w_str(p,"\P\\fs");
      NumToString((long)p->att->bSize << 1,&string);
      w_str(p,&string);
      if (p->att->bStyle & bold) {
         w_str(p,"\P\\b");
      }
      if (p->att->bStyle & italic) {
         w_str(p,"\P\\i");
      }
      if (p->att->bStyle & underline) {
         w_str(p,"\P\\ulw");
      }
      if (p->att->bStyle & outline) {
         w_str(p,"\P\\outl");
      }
      /* Output ruler stuff */
      if (p->level > 1 && p->labelType == DECIMAL)
         i = 48;
       else
         i = 32;
      if (p->labelType == BULLETS) i = 16;
      if (p->labelType == NONE) i = 0;
      /* Output first indent and left indent in twips (1/20
      point) */
      w_str(p,"\P\\li");
      NumToString((long)(p->level * 16 + i) * 20,&string);
      w_str(p,&string);
      w_str(p,"\P\\fi");
      NumToString((long)i * -20,&string);
      w_str(p,&string);
      w_byte(p,' ');   /* Be sure not to run into text */
                       /* Output label */
      if ((p->label)[0] != 0) {
         for (i = 1; i <= p->label[0]; i++)
            w_byte(p,p->label[i]);
      }
      if (p->att->bType == PICTtype) {
         w_str(p,"\P{\\pict\\macpict\\bin");
         NumToString((long)p->len,&string);
         w_str(p,&string);
         w_byte(p,' ');
         count = p->len;
         error = FSWrite(p->fRef,&count,p->data);
         w_byte(p,'}');
       } else {          /* Not a PICT */
         warp(p,p->len,p->data);
      }
      w_str(p,"\P\\par}\r");
      break;
    case 1:
      w_str(p,"\P}\r");
      break;
   }
   return(p->already_error);
}

/*
FILE_ERROR - Display the appropriate message for a file error
*/
void file_error(error,p) register OSErr error; P_PTR p; {

   if (p->already_error != noErr) return;   /* Only once */
   p->already_error = error;      /* Indicate error */
}

/*
WARP - Escape {,\,},CR,TAB
*/
void warp(p,len,s) P_PTR p; long len; register char *s; {
   long      count;
   register  word   i;
   register  char   c;

   if (p->already_error) return;  /* Already have error */
   for (i = 0; i < len; i++) {
      c = *s++;
      switch (c) {
        case '{':
        case '}':
        case '\\':
         w_byte(p,'\\'); w_byte(p,c);
         break;
        case CR:
         w_str(p,"\P\\line ");
         break;
        case TAB:
         w_str(p,"\P\\tab ");
         break;
        default:
         w_byte(p,c);
      }
   }
}

/*
W_BYTE
*/
void w_byte(p,b) P_PTR p; unsigned char b; {
   long                 count;
   register OSErr       error;

   if (p->already_error) return;     /* Already have error */
   count = sizeof(char);
   error = FSWrite(p->fRef,&count,&b);
   if (error) {
      file_error(error,p);
      return;
   }
}

/*
W_STR - Write a Pascal string
*/
void w_str(p,s) P_PTR p; char s[]; {
   long                 count;
   register OSErr       error;

   if (p->already_error) return;     /* Already have error */
   count = s[0];
   error = FSWrite(p->fRef,&count,&s[1]);
   if (error) {
      file_error(error,p);
      return;
   }
}
```

«listing 3»

Listing 4 summarizes the additional resources which make up an output driver.

```
resource 'BNDL' (128) {
    'Artf',
    0,
    {   /* array TypeArray: 2 elements */
        /* [1] */
        'ICN#',
        {   /* array IDArray: 1 elements */
            /* [1] */
            0, 128
        };
        /* [2] */
        'FREF',
        {   /* array IDArray: 1 elements */
            /* [1] */
            0, 128
```

```
        }
    }
};

resource 'FREF' (128, "a_TEXT") {
    'ACTF',
    0,
    ""
};

data 'TYPE' (0, purgeable) {
    'TEXT'
};

data 'CREA' (0, purgeable) {
    'MSWD'
};

data 'RULR' (0, purgeable) {
    $"0000"
};

data 'Artf' (0, purgeable) {        /* Signature */
    "©1987 Maitreya Design 11 March 1987"
};
```

and the ICN#s

«listing 4 — *Rez* input»

Doing it with Pascal

It's a little known fact that *Acta* makes a superior Scrapbook. The example in this section is a relatively simple input format driver (in *Lightspeed* Pascal [*LSP*]) which reads a scrapbook file into an *Acta* outline, converting each text block and picture in the scrapbook to a separate topic. With this driver, those of you with a lot invested in scrapbooks can painlessly convert to *Acta*.

Unfortunately (for Pascal programmers) *Acta* assumes its format drivers are C functions. This is because *Acta* is written in C, and because David takes a perverse pleasure in making Pascal programmers read C declarations. (The authors of ***Inside Macintosh*** no doubt have a similar feeling.) But some Pascal compilers won't generate code that can call or be called by code written in C. This section will discuss a relatively general method of coercing a standard Pascal compiler to generate code compatible with C. Note that *MPW* Pascal allows you to declare procedures as C functions so they can talk directly to *Acta*.

Linking Code Produced by Different Compilers

Except for a few systems like *MPW*, it's not possible to link code compiled by different compilers. On the Macintosh, one way to overcome this is to compile code into code resources. These can then be loaded using the segment loader, or in the case of a DA such as *Acta*, with GetResource. Using this method it's possible to separately compile and link blocks of code totally independent of which compiler generated them, using virtually any compiler desired by the programmer.

Argument Order

In both C (at least *Aztec C*, in which *Acta* is written; *MPW* C, used in the examples; and *LightspeedC*, another compiler we're familiar with) and Pascal, arguments are passed to procedures on the stack. However the order in which the arguments are pushed onto the stack is different for the two languages. Pascal pushes arguments onto the stack in the order in which they are declared. C pushes the arguments in the reverse order of declaration. (Note that this allows C routines to have a variable number of arguments; the first argument is always in a known place [the top of the stack] and can indicate how many other arguments there are.)

If a Pascal procedure is to be callable from C, it must declare its arguments reversed from the C function call, for example:

The C declaration for the format driver main procedure read_Scrap is:

```
OSErr read_Scrap(infil,resfil,att,add_topic)
word infil, resfil; BLOCK_ATT *att; ProcPtr add_topic;
```

while the Pascal declaration is:

```
Function read_Scrap(add_topic:ProcPtr; att:Block_Att;
resfil,infil:Integer) : OSErr;
```

When calling a C function from Pascal the arguments must be passed in the reverse order from the C function declaration, for example:

If the C declaration for add_topic is:

```
void add_topic(data,length,level,attributes,styleHandle)
Ptr data; long length; word level; BLOCK *attributes; Handle
styleHandle;
```

the Pascal call to the function add_topic is:

```
add_topic(Nil,att,level,size,^text);
```

while the C call is:

```
(*add_topic)(*text,size,level,att,NIL);
```

Argument Passing

C allows a variable number of arguments to be passed to a function, so the function can't simply remove a fixed number of bytes from the stack before returning as is done in Pascal. When a C function calls a Pascal procedure (*Acta* calls the a_Scrap main procedure) this creates problems as both the Pascal procedure and the calling C function will remove the arguments from the stack. The converse problem occurs when a Pascal procedure calls a C function (a_Scrap calls add_topic) — no one removes the arguments from the stack.

For some Macintosh Pascal compilers (*Lightspeed* and *TML*) this problem can only be solved by resorting to assembly language "glue" routines. Those of you who don't care for assembly code please just hold your nose and read on. The result of this discussion is two small, relatively general assembly procedures. One procedure lets *Acta* call an input format driver written in Pascal and the other allows a Pascal driver to call *Acta*'s add_topic procedure. You won't have to write any assembly language to add your own input format driver (well, as long as the format driver interface doesn't change anyway), though output format drivers are left as an exercise for the reader (or a future column)!

Calling a Pascal procedure from C involves declaring the Pascal procedure as having no arguments, and copying the arguments from the stack into local variables using an assembly procedure (Listing 5). By doing this Pascal will not mess with the stack by deleting the arguments. This is one of those areas where languages like C have it all over Pascal, which was designed to

keep the programmer as far from the machine as possible. This promotes machine independence, but like most blessings extracts a price. When you absolutely must get down to the bare metal of the machine, you're forced to use assembly language to do it.

```
;  Procedure GetArguments( VAR add_topic_Ptr : ProcPtr;
;            VAR att : Topic_Attributes;
;            VAR ResRefNum : Integer;
;            VAR Refnum : Integer);

This procedure will pull the arguments passed to input
routines from the stack and put them in the input routines
internal variables.  This procedure will not change the stack
from what C expects on return from the input routine.

This procedure is called as the first statement in the input
module.  it assumes that A6 was linked at the beginning of the
code module.  this occurs if there are any local variables at
the top level of the code module main entry point routine.

;
;  Stack looks like this:
;  ---------------------
;  input parameters to input module from Acta
;  ---------------------
;  Return Address
;  ---------------------
;A6===>   Old A6
;  ---------------------
;  arguments for GetArguments
;  ---------------------
;  GetArguments calling procedure return address
;  ---------------------
;A0===>   Old A0                    <==== SP
;  ---------------------
;
;
;
   link      A0,#0     ; A0 points to getarguments stack frame
   move.L A2,-(SP)     ; save A2
   move.L 8(A0),A1     ; A1 = ptr to RefNum
   move.W 8(A6),(A1); move refnum from c frame to pascal frame
   move.L 12(A0),A1    ; A1 = Ptr to ResRefNum
   move.W 10(A6),(A1)  ; move ResRefNum from c to pascal frame
   move.L 16(A0),A1    ; A1 = Ptr to ATT
   move.L 12(A6),A2    ; A2 = ptr to C ATT
   move.L (A2)+,(A1)+  ; move C ATT to pascal ATT
   move.L (A2)+,(A1)+
   move.L (A2)+,(A1)+
   move.L 20(A0),A1    ; A1 = ptr to Pascal Procptr
   move.L 16(A6),(A1)  ; move procptr from c to pascal
   move.L (SP)+,A2     ; restore A2
   unlk      A0
   rts
```

«listing 5»

Function Results

Pascal allocates space on the stack for the functional result and places the arguments on the stack. When the function returns the result is on top of the stack. A function written in C returns the functional value in register D0. *Acta* (a C program) calls input format drivers as functions. A format driver written in Pascal must conform to this standard for passing functional results.

A Pascal format driver must be defined as a procedure so it doesn't try to write its function result to the stack. An assembly language procedure is called as the driver's last statement to load the functional result into register D0 (Listing 6).

```
;  Procedure CFunctionReturn(Error:OsErr);
;
This procedure is a kludge to allow C and Pascal routines to
work together Acta calls the a_Text Main routine and expects
its functional value to be in the register D0, however Pascal
expects there to be space reserved on the stack for it, and
would save it there if allowed. This procedure takes an
argument which would normally be set equal to the functional
result and puts it in D0.  The calling procedure will then not
set the function equal to this value.

   move.L (SP)+,A0   ; Get Return address
   move.W (SP)+,D0   ; put functional result in D0 for C
   jmp    (A0)       ; RTS
```

«listing 6»

Calling a Procedure Passed as an Argument

Acta passes the address of add_topic as an argument to the input module. Unfortunately Pascal can't call a procedure by pointer directly. The solution, once again, is an assembly glue routine to take this procedure address as an argument and jump to the procedure (Listing 7).

```
; add_topic(text: Ptr;size: LongInt;level:Integer;
;  att:Topic_Attributes; add_topic_Ptr:ProcPtr);
;

This procedure is the glue routine between pascal call to
add_topic and Acta's add_topic routine.  Pascal can not call a
procedure by using the procptr so a ASM glue routine is
required.  In addition C requires the calling routine to take
its own arguments off the stack. The procptr was added to the
add_topic argument list on the end so that this routine could
easily pull it off and be left with the arguments on the stack
as C wants to see them.  This procedure must then remove the
arguments off the stack as the C procedure will not do it.

;
add_topic
   lea    SaveReturn,A0
   move.L (SP)+,(A0)    ; save return
   move.L (SP)+,A0      ; get subroutine address
   jsr    (A0)          ; JSR to C add_topic routine
   lea    18(SP),SP     ; pull arguments off stack
   lea    SaveReturn,A0 ; restore return address
   move.L (A0),-(SP)
   rts
SaveReturnDC.L   0   ; storage for return address
```

«listing 7»

Putting It All Together in Pascal

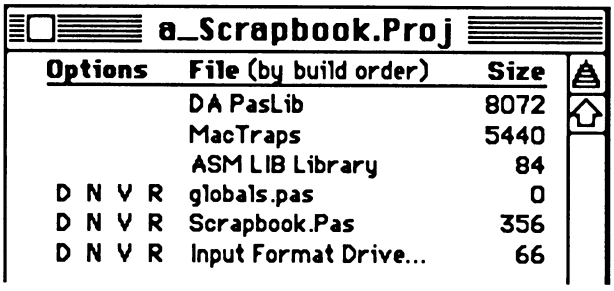
Listing 8 is a shell which can be called from *Acta*. A call to the user's input driver is inserted in this shell. The user's format driver code is stored in a separate unit and a uses clause which references it is added to the shell replacing the Scrapbook Uses. The assembly language procedures are contained in a file called ASM LIB Library which is compiled using Consulair's *Assembly Development System* (formerly *MDS*) and converted into an *LSP* library using *REL Convert*. This library must be added to the project in build order before the format driver code (see figure). See the *LSP* manual for more information on how to build a code resource.

```
UNIT InputDriverShell;
```


© The Definitive MacTutor, Vol. 4

```
INTERFACE
   USES
      Globals, Scrapbook;

   { main entry point for the code unit }

PROCEDURE Main;

IMPLEMENTATION

PROCEDURE Main;

      VAR
         anErr           : OsErr;
         add_topic_Ptr : ProcPtr;
         att             : Topic_Attributes;
         ResRefNum       : Integer;
         Refnum          : Integer;

  { def for ASM C/Pascal glue for Acta input format driver }
   PROCEDURE CFunctionReturn (anErr : OsErr); external;
   PROCEDURE GetArguments (VAR add_topic_Ptr : ProcPtr;
                           VAR att : Topic_Attributes;
                           VAR ResRefNum : Integer;
                           VAR Refnum : Integer); External;

   BEGIN

      { get the funtion arguments from the stack,
         behind compiler's back }
      GetArguments(add_topic_Ptr, att, ResRefNum, Refnum);

      { call the specific input format driver }
      anErr := a_ScrapBook(add_topic_Ptr, att,
         ResRefNum, Refnum);

      { put the functional result in register D0 so
         Acta can get it }
      CFunctionReturn(anErr);

   END;
END.
```

«listing 8»



Finally, here's the actual source code which converts TEXT and PICT resources in a scrapbook file (actually, in any resource file) into *Acta* topics.

```
UNIT Scrapbook;
{ this unit will read a scrapbook file into a set of Acta
topics for use with Acta 2.0 or higher }

INTERFACE

   USES
      Globals;  { Pascal def of Topic_Attributes record }

   FUNCTION a_ScrapBook (add_topic_Ptr : ProcPtr;
                         att : Topic_Attributes;
                         ResRefNum : Integer;
                         Refnum : Integer) : OsErr;

IMPLEMENTATION

   FUNCTION a_ScrapBook;

      CONST
         ActaTextTopic = 1;
         ActaPictTopic = 2;
         TopicLevel1 = 1;
         TopicLevel2 = 2;

      VAR
         ResHandle : Handle;
         TempPtr : Ptr;
         anErr : OSErr;
         ErrorString : Str255;
         RError : Integer;
         AppResNumber : Integer;
         ResCount : Integer;
         Size : LongInt;
         TType : OSType;
         i, j, k : Integer;

   { Acta internal procedure for adding a topic to the active
window. This procedure was written in C, so arguments must be
put on the stack in the reverse order of Dave's documentation.
This procedure is really linked to an ASM glue routine which
pulls the pointer to the add_topic procedure off the stack and
jumps to it and pulls the arguments off the stack before
returning as well }

   PROCEDURE add_topic (empty : Ptr;
                        att : Topic_Attributes;
                        level : Integer;
                        size : LongInt;
                        text : Ptr;
                        add_topic_Ptr : ProcPtr);external;


   PROCEDURE Init;

   { initialize a_Scrapbook   }

         VAR
            Watch : CursHandle;

   BEGIN

      Watch := GetCursor(WatchCursor);
      { change cursor to watch while converting file }
      SetCursor(Watch^^);

      { save away the current resource file }
      AppResNumber := CurResFile;

      { Read from the input file }
      UseResFile(ResRefNum);

      { start out with the text blocks }
      ATT.bType := ActaTextTopic;
      TType := 'TEXT';

   END; { Init }

BEGIN { a_Scrapbook }

   Init;      { Initialize }

   FOR j := 1 TO 2 DO
      BEGIN
      { if the current output type is text then count TEXT }
      {  resource, else count number of picture resources }
         ResCount := CountResources(TType);
```

```
        RError := ResError;
        IF RError = noErr THEN
           BEGIN
           { loop through all resources of current type }
           FOR i := 1 TO ResCount DO
              BEGIN
              ResHandle := GetIndResource(TType, i);
              RError := ResError;
              { convert only resources actually in
              the scrapbook file }
              IF (RError = noerr) AND (HomeResFile(
ResHandle) = ResRefNum) THEN
                 BEGIN
                 HLock(ResHandle);
                 Size := Longint(GetHandleSize(ResHandle));
                 add_topic(NIL, ATT, TopicLevel2, Size,
ResHandle^, add_topic_ptr);
                 HUnLock(ResHandle);
                 END; { If (RError = ...}
                 ReleaseResource(ResHandle);
              END; { For (rescount)  }
           END; { RError)
        ATT.bType := ActaPictTopic;
        TType := 'PICT';
     END; { for (1 to 2) }
     { Use the application resource fork }
     UseResFile(AppResNumber);
     InitCursor;       { initialize cursor }
     { somewhat simplistic error reporting }
     a_ScrapBook := RError;
  END; { a_ScrapBook  }
END. { ScrapBook }
```

«listing 9»

Conclusion

Format drivers allow programs to be extended easily, adding new features without an entirely new program. They also allow the user to customize the program's features. On the implementation side, they are easy to write and allow packages to be written in multiple languages, even with development systems that do not normally support this.

Extensibility need not be limited to handling file formats. Future versions of *Acta* may have feature drivers.

The shrewd reader will have noticed that the techniques we've discussed allow you to break up your program into pieces which can be sold separately. This means you can get the program out the door quicker, and add the features later. And your marketing experts can have a field day packaging the different pieces.

Exercises

The answers to these exercises will not appear in next month's ***MacTutor***. But, you may be able to release them as free or shareware programs.

1. Modify a_TEXT to support *MPW*'s font and size specification.

2. Write an input driver which reads a tab (or comma) delimited file, creating a new level at each delimiter. (This format driver should have a name such as a_TEXT/tab, so the user has the choice of how to split topics.)

3. Write an input driver which reads a text file into a single topic. *Acta* uses TextEdit, so check for files which are too large. (This format driver should have a name such as a_TEXT/1 topic, so the user has the choice of how to split topics.)

4. Adopt the format driver technique in your own program.

5. (extra credit) Write a format driver for Microsoft *Word* 3.0 format.

Appendix — Scrolling Icon List

Implementing the user interface really isn't in the scope of this article, but we haven't seen many examples of custom list definition functions (LDEFs).

Adding a scrolling list to the Standard File dialogs is pretty easy. Custom (SFPGetFile/ SFPPutFile) dialogs have a userItem, and the List Manager handles the work (with the aid of a custom LDEF). The dlgHook initializes the list when it gets passed item -1, and the filterProc checks for mouseDown events in the list. The only tricky part is in disposing of the list after the SFPPutFile call. If you call LDispose after the call, the List Manager tries to call DisposControl for the scroll bar...but DisposDialog has already disposed of all the controls. If you try to call LDispose in the dlgHook after a click on the OK button, you're faced with the problem that the dialog may not really go away — the user may decide not to replace a file with the same name. Our solution was to set the list's vScroll field to NIL before calling LDispose, since after the return from SFPPutFile, the control no longer exists.

Flying in the face of ***Inside Macintosh*** volume IV, the LDEF is written in C. The routine presented here shows only how to draw labelled icons (*Acta*'s LDEF uses additional data structures, and disposes them when it receives an lCloseMsg).

```
struct ListEntry {
    Handle   icon;
    char     name[64]; /* Icon label (file name) */
};
typedef struct ListEntry   ListEntry;

/*
LDEF - List Definition Procedure
*/
pascal void
ldef(message,select,rect,cell,dataOffset,dataLen,handle)
word              message;
word              select;
Rect              *rect;
Cell              cell;
word              dataOffset;
word              dataLen;
ListHandle        handle;
{
    Rect          box;
    ListEntrylistData;
    BitMap        iconBits;
    GrafPtr       thePort;
    char          displayName[64];

    if (dataLen == 0) return;  /* Nothing to draw (excludes
```

```
                                lCloseMsg) */
    GetPort(&thePort);/* Get current grafPort */
    /* Get our rectangle (only lDrawMsg, lHiliteMsg should get
    past */
    /*  the dataLen test) */
    box = *rect;  /* Make a copy so we can change it */
    box.top += 2;
    box.left += 16;
    box.bottom = box.top + 32; /* Icons are 32 pixels */
    box.right = box.left + 32;
    /* dataLen is already set */
    LGetCell(&listData,&dataLen,cell,handle);
    switch (message) {
     case lDrawMsg:
        EraseRect(rect);
        PlotIcon(&box,listData.icon);
        /* If name begins with "a_" then ignore prefix */
        if ((listData.name[1] == 'a' || listData.name[1] ==
'A') &&
            listData.name[2] == '_') {
            /* Make a copy, since listData has filename */
            displayName[0] = listData.name[0] - 2;
            BlockMove(&listData.name[3],&displayName[1],(long)displayName[0]);
         } else {
            BlockMove(&listData.name[0],&displayName[0],
                (long)listData.name[0] + 1L);
        }
        TextFont(geneva);
        TextSize(9);
        MoveTo(rect->left + 32 - StringWidth(displayName) /
2,
            rect->top + 32 + 12);  /* Center the name */
        DrawString(displayName);
        /* Restore typestyle to assumed value */
        TextFont(systemFont);
        TextSize(12);
        if (!(select & 0x100))
            break;
        /* else fall through */
     case lHiliteMsg:
        /* Set up the bit map */
        iconBits.rowBytes = 4;
        SetRect(&iconBits.bounds,0,0,32,32);

        /* According to Steve Brecher, no need to lock, since
        CopyBits won't alter the heap configuration
    unless a picture or region is being recorded. */

        iconBits.baseAddr = (char *)(*listData.icon) + 128;
        CopyBits(&iconBits,&thePort->portBits,
&iconBits.bounds,&box,
            srcXor,NIL);      /* Blit the icon */
        break;
    }
}
```

Acta Document Format

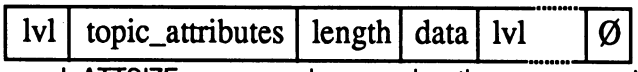
Written by David Dunham
©25 September 1987 Maitreya Design
Maitreya Design
POB 1480
Goleta, CA 93116

This note assumes you are familiar with *Acta* and its terminology. *Acta* documents have the type 'OTLN' and creator 'ACTA.'

An outline on disk consists of a data fork. This is simply the data for each topic in sequence (top-to-bottom order). The file is terminated with a zero word.

Each topic is preceded by its level. Topics at the left margin are level 1, their daughters 2, etc. The topic's attributes indicate its type and formatting. Data is preceded by its length as a long-word. The data itself is either text (not a Pascal string!) or a QuickDraw picture, and is not padded.

The same format is used for the clipboard; *Acta* uses a scrap type of 'ACTA.' (It also writes a TEXT scrap.)




```
struct TOPIC_ATTRIBUTES {
    word  bType;            /* TEXTtype, PICTtype */
    word  bAttributes;      /* currently 0, except for
                               optionWord bits */
    word  bFont;            /* Typestyle */
    word  bStyle;
    word  bSize;
    byte  bColor;           /* color index */
    byte  bExtra;           /* reserved; currently 0 */
    byte  firstLine;        /* Show first line only? */
    byte  expanded;         /* Y/N */
    word  hidden;           /* Is block hidden (by Collapse)?
                            count of hide depth */
}
```

BType is 1 for a text topic, 2 if the topic is a QuickDraw picture. Other values are reserved for future use. BAttributes is currently unused and should be left 0, except that the bAttributes for the first topic is used to determine the options. BFont, bStyle, and bSize determine the typestyle of a topic, and have their usual QuickDraw interpretation. BColor is an index into a table of color values; 0 is black. BExtra is reserved and should be 0. First-Line should be set to TRUE (-1) if the topic is shrunken, FALSE (0) otherwise. Expanded is TRUE if the daughters are visible, FALSE if they are hidden. If a topic is visible, hidden is 0, otherwise it is the number of collapsed ancestors.

If bAttributes of the first topic is 0, the default options of alphabetic sort A-Z, no ancestress constraint, and outline current topic are used. You can choose different options by setting the following bits (all other bits should be left reset). These bits are ignored if set on topics other than the first.

```
0x0100  numeric sort
0x0001  Z-A sort
0x0002  single ancestress
0x0004  "smart quotes"
0x0020  don't label pictures
0x0040  don't outline current topic
0x0080  don't put bullets in Clipboard
```

In future releases, additional information (such as TextEdit 'styl' records) may be added. It will follow the zero word. This data can be ignored, as the above format will be used for backward compatibility.

Pascal Procedures

MultiFinder Friendly MacDraw Plotter

Dave Kelly
MacTutor Editorial Board
David Smith
Editor & Publisher
MacTutor Vol. 4 No. 2

For the past few months, MacTutor has been running a series of editorials on the importance of developers getting with the new way of doing things. In the last six months, Apple has introduced a number of product enhancements that directly affect commercial software development. These include:

- MultiFinder Quasi-Multitasking Finder
- AppleShare File Server
- New Menu Manager with nested menus
- New Color PICT format
- New Color Quickdraw Capability
- New Style-enhanced Text Edit & Clipboard type
- Multiple screen capability
- Universal Hyper-Text type data base
- Faster number crunching with 68881 chip
- New ROM interface files, MPW 2.0
- Lots of new Technical notes fixing old bugs
- New compatibility headaches

As usually happens when new goodies come on the market, everyone looks at their old software and says "how come my version of MS____ doesn't support this?" As programmers and developers, we have a responsibility to bring our software up to speed as quickly as possible with the new way of doing things. For example, the time-consuming dialog box to change text font and style is no longer good enough when nested menus can do the same job so much faster. Black and white windows, menus and documents get old after using color for a few weeks. Big screen monitors make old style paint programs frustrating when you can't grow the window, or have to turn off color to keep the program from crashing. And program development environments that once were speedy now send the Mac into the bit-bucket because the developer can't figure out how to handle the switching context with a color video card installed. Even lowly debuggers have problems trying to cope with all the system enhancements. So while Apple has indeed given us lots of new toys, they have also given us lots of new headaches trying to understand and upgrade our products to work with everything. If you thought it was hard to program the Mac two years ago, pity the programmer just getting started today. And the fact that Apple has just now managed to cleaned up the documentation in *Inside Macintosh Volume 5*, doesn't help much.

Put Up or Shut Up

After hi-lighting commercial products which have made the transition to the new rules of the game, we have decided to put our money where our editorial mouth is, and attempt to publish a program that illustrates how to incorporate many of the new system features Apple has introduced in the last six months.

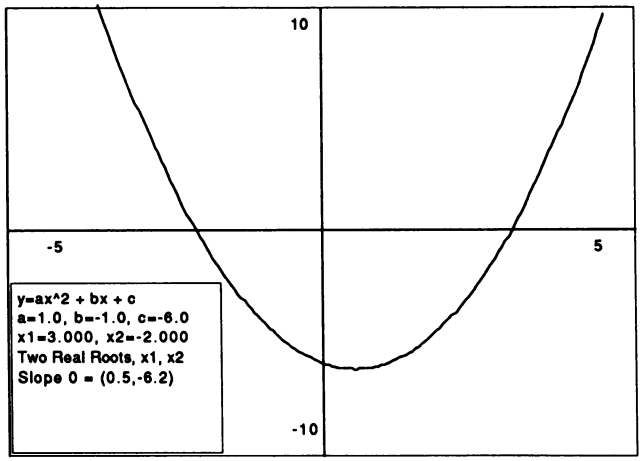


Fig. 1 Plotter Program generates hairline plots placeable directly in Pagemaker.

Problem Solving

Since one of the hottest markets for Macintosh now is in engineering environments, we will discuss a typical engineering approach to computer use. In aerospace and the defense industry, the name of the game is *problem solving*. You use the computer to solve a problem. It may be physics, engineering or math, but the computer is the tool for investigating the problem and finding a solution. Let's suppose our problem is to build a space shuttle that flies rather than blows up. And let's suppose a key part of the problem is the equation for modeling the burn rate of the solid rocket fuel in the boosters. To make things simple, we will assume the modeling equation is quadratic in form. We want to use the computer to model this quadratic equation and all others like it so we can determine the best model for the rocket's design. The problem most engineers have when trying to use the Macintosh for problem solving, is they spend so much time figuring out how the Mac works, they never get around to the problem! So here is a shell program for modeling and simulation programs that follows the new Macintosh way of doing things. Our "Plotter" program analyzes the quadratic equation and plots it with the following Mac features:

- Dialog box for user-friendly input of parameters
- Equation plotted as a Quickdraw picture
- Plot can be saved to disk as a PICT file
- Program supports printing:
  - hairline line width for graph
  - Choice of window size or paper size
- Plot Window is growable, zoomable, moveable
- Plot can be opened and edited in MacDraw
- Plot can be placed in Pagemaker
- Uses Quickdraw and Postscript comments

© The Definitive MacTutor, Vol. 4

- Supports WaitNextEvent for MultiFinder use
- Quickdraw colors for menus, graph, axis
- Nested menus allow color changes

Mac Modeling Requires Quickdraw

One of the big problems in trying to model equations on the Macintosh is how to create a Quickdraw Picture. Naturally you could just draw it on the screen as a paint-type bit-map, but then you couldn't take advantage of the higher resolution of the LaserWriter to show the details of the plot. Once a plot is created, you want to be able to save and edit it to add notes, or even move the graph around the axis. To do this, the parts of the plot have to be created as MacDraw type objects. Yet they also have to be grouped or else you end up with 200 date points being re-positioned on the page by a careless mouse. When the plot is finished you have to be able to print or import it into a program like Pagemaker to show it to someone. So how to you create a program that does all this?

Check Your Library!

First, you make sure your supply to Technical Notes is up to date. Many of these features are documented and supported by the just released *Volume Five of Inside Macintosh*, and by a rash of new Technical Notes released since the last Boston Expo. Here are some of the notes that deal with the problems of pictures from the point of view of modeling equations.

- 21 Internal Picture format 11/86
- 27 MacDraw Picture Format 9/86
- 59 Pictures and Clip Regions 1/86
- 72 Optimizing for the Laser 3/86
- 91 Picture Comments 3/87
- 152 Using Laser Prep routines 7/87
- 154 Displaying large PICT files 7/87
- 155 Handles & Pointers 9/87
- 181 Picture Comments 11/87
- 183 Position Independent PS 11/87

In addition, there is the *LaserWriter Technical manual*, a new *PICT File Format Notes*, and *Mac Example Applications, ver. 1.0*. These three publications, available from APDA, include a wealth of information on how to create and print Quickdraw pictures and pic comments, both MacDraw and Postscript comments.

The problem of making an application MultiFinder friendly is partly addressed in the following technical notes as well as the recently released *MultiFinder Development Package, ver. 1.0*, from APDA:

- 158 Asked MultiFinder Questions 9/87
- 177 Sleep Parameter problem 11/87
- 180 MultiFinder Miscellanea 11/87

One of the best "overview" publications of how to implement all of the new goodies, especially color, is Dave Wilson's new note collection, *Programming the Macintosh II*. This is available from MacTutor's mail order store, in this issue. Dave has collected the slides from his Mac II seminar at last year's Boston Expo and published his color paint program source code to provide a nice one-stop shopping center for all of the features in our program this month. Finally, you should plan on getting the *Professional Programmer's Extender* from Invention Software. This contains all of the source code for volumes 1 and 2 of their extender products. This is also available from the MacTutor mail order store, in this issue. It is much faster to cut and paste source code from someone else, then re-invent the wheel yourself.

For those who like books (and have the time to read them), your in luck, if you use Borland's Turbo Pascal for the Mac. A rash of new books have been released including Macintosh Turbo Pascal by Tom Swan [Wiley & Sons] and Turbo Pascal for the Mac by Leon Wortman [Tab Books]. Borland has also released a bunch of goodies including *Turbo Pascal Tutor*, and *Numerical Methods Toolbox*. I make it a habit to just visit my local computer bookstore regularly and buy anything with Mac on it. Call it a business expense!

How to Use the New Goodies

Our program this month features the full Macintosh User Interface, as documented in the recently released *Human Interface Guidelines: The Apple Desktop Interface*, published Addison-Wesley, for Apple. It would be a great challenge to take this book as a program definition and attempt to write a demo program that included full support for every feature in this book, showing how the full User Interface is implemented. The main program, and a unit called MyPlotStuff, implement the standard event loop for a single window application. Much of this was cut from the Multi-Window Text Editor MacTutor published last year. As I said, its always faster to cut and paste than re-invent. To support color, a color array is set up in the InitMac routine to define the eight Quickdraw colors. This way the program can run on all Macs without a lot of special programming. (Apple really needs to put color quickdraw into the SE so that all machines have the same version of Quickdraw, even if they can't show color.)

MultiFinder Friendly

To be considered MultiFinder friendly, an application has to call WaitNextEvent and support suspend and resume events correctly. Our program works under MultiFinder and is especially useful with MacDraw, saving and opening our graphs between the two programs. But it wasn't easy figuring out how to do it. The first problem we had was that neither Think's Lightspeed Pascal nor Borland's Turbo Pascal support the final MPW 2.0 interfaces. The latest versions, (LS Pascal 1.11 and Turbo Pascal 1.1) both use MPW 2.0 beta interfaces and so do not include either the SysEnvirons glue, nor the WaitNextEvent interface. To get around this, I added the WaitNextEvent definition to the ROM85 library for LS Pascal. Since SysEnvirons is an OS trap, I suspect you can't call it without assembly glue to manage the register set-up so it is commented out in the listing. I just assumed the Mac was a Plus, SE or II with the latest system.

Where Is WaitNextEvent

The MultiFinder documentation says you should call WaitNextEvent if it is present. Otherwise, call GetNextEvent and SystemTask. So you have to check the unimplemented trap call to find out and set a flag so you will know which event trap to use. I did this and after a while I noticed the program always said the WaitNextEvent trap was unimplemented. What? I have a Mac II with the latest system. How can it be unimplemented? Where is it? I called several people, both at Borland and Think, but no one

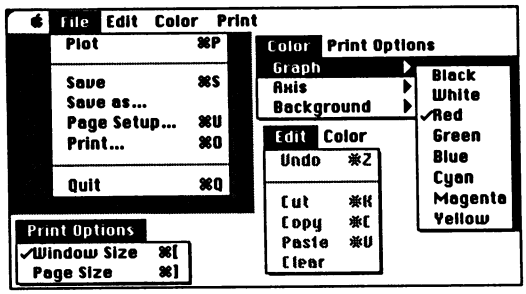


Fig. 2 Our menus include a nested color menu

knew the answer. Finally, after studying two sample programs from the MultiFinder Developer's package previously mentioned, I noticed their programs didn't seem to have any problem. It finally dawned on me. WaitNextEvent is inside MultiFinder! Thus it only exists when your application is running under MultiFinder, when I suppose it gets patched into the system. Under the Finder, it's not there so you have to check for it. You would think this little gem would have been included in the documentation somewhere. Alas, I don't have either the final Volume 5 of IM, nor the final MultiFinder Developer's package!

Our plot is stored as a picture in a data structure called PlotDocHandle. We set up the document in our InitMyWindow routine in the normal manner. Note that all the rectangles we will need are set here, relative to the window's portRect. We also do a ClipRect to set the clipping region of the window's grafport so we don't have any Quickdraw problems a la tech note 59.

Color Menus

Our InitMyMenus routine includes an apple, file and edit menu as well as a nested menu for color and a menu for some printing options. The data structures are designed to make it easy to check the nested menu items and this code depends heavily on both Dave Wilson's programming examples, and other code segments I've seen.

The Mighty Update Event

Our doMouse, Keydowns, and activate routines are pretty standard user interface stuff so no need to go over those here. The Update routine, is critical. The ideal for all Macintosh programs is to never draw anything! The application creates a Quickdraw picture or pictures and then invalidates some portion of the screen, causing an update event. Then, in the Update Event routine, you call DrawPicture to display the results of all the other application code that has been maintaining and modifying those pictures. It has always been a mystery to me how you create the perfect Update Event routine. Here is how ours works:

- Save and set the current grafport
- Begin Update to restrict the visRgn to the collection of invalid rgns that need updating.
- Erase the window's portRect.
- Set the clip rgn to the window's portRect.
- Draw the picture.
- Draw the grow icon.
- End the update, restoring the VisRgn.

Exactly why all of these steps are needed, in that order is a mystery to me, but it works. The window's portRect encloses the content region of the window, which includes everything but the title bar. In particular, it includes the scroller and grow icon regions. So when you erase that, you have to re-draw the grow icon. The picture is drawn in what I call the PicRect, or the application area of the content region, not including the scroller areas. When a grow window event takes place, these rectangles have to be carefully updated so that the next update event will work properly.

MultiFinder Support

Our doMulti routine implements the suspend and resume events from MultiFinder. Basically, it is a carbon copy of our activate / deactivate event routine. Since it is not really an event, we have to do a lot of processing to check for suspend or resume, and for a mouse moved event, if we are supporting that feature of MultiFinder. Our program is set up for it, but we don't support it, since, for multiple monitors, you have to do a lot of region arithmetic I don't fully understand. The beta documentation also says we have to call SystemEvent and post an activate event if a DA is the front window when we are suspended or resumed. I have included this code in our doMulti routine, but it may not be necessary. I'm looking forward to checking the final documentation to see about this.

Grow Window

Our MyPlotStuff unit implements the remainder of the usual user interface. We have an about box dialog that reads the ID string like the new Finder does and displays it. We also find out how much memory we have left from whatever MultiFinder has allocated to us by calling MaxMem. Our doMenuBar routine does the usual menu bar processing, including checking the nested color menu items when the user selects a new color. The menus also are displayed in the color of the selection thanks to the color menu resources Steve Sheets taught us about in past issues of MacTutor.

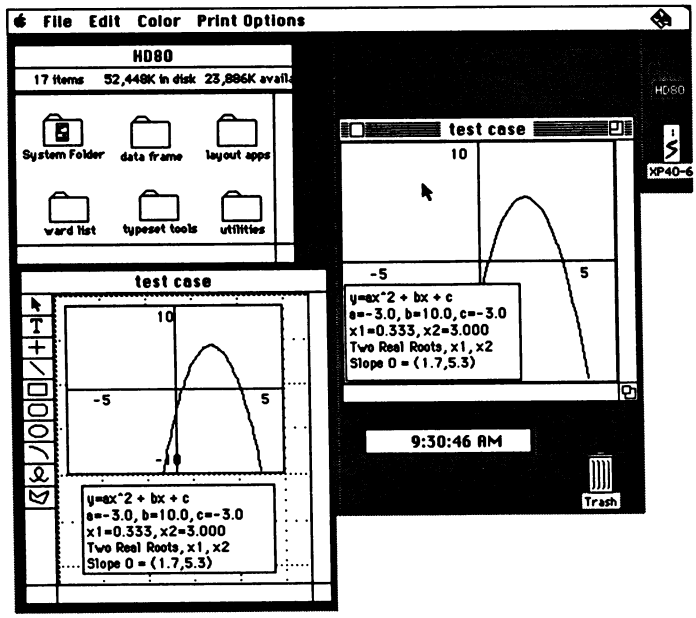


Fig. 3 MultiFinder support: Plotter works with MacDraw

Our grow routine is the most important one in this unit. Grow works with the update event to change the window properly when it is resized. It is another Macintosh mystery I've never fully understood. As I understand it, you have a pre-grow period and a post-grow period, determined by when you call SizeWindow. I use the routine for both a grow window event and a zoom window event. In the case of a zoom, you get the new window size from the portRect, which is already set for you by the zoom. In the case of a grow, you call GrowWindow to return the new window size. I re-calculate the important window rectangles both before a SizeWindow and after. SizeWindow changes the portRect of the window to the new size. So, first you invalidate the scroller, grow and pic rectangles with the old portRect, then you SizeWindow to change the portRect, and invalidate the scroller, grow and pic rectangles for the new window size. This seems to correctly get all the window areas properly drawn during the update event. I suspect we may be invalidating to many areas and that this can be re-fined so that you only invalidate certain rectangles depending on whether the window is getting smaller or larger. Also, if you have controls, you can move and re-size controls separately, so they can be removed from the invalidation by calling the validate trap. If anyone has a more efficient grow/update routine for the general case, I'd like to see it.

Saving Our Graph

Once we have played around with various graphs, it would be nice to save them. Our myFileStuff unit does the SaveAs and Save functions. To save a picture, you call the standard file dialog and if the reply is good, you create the file, open the file, set the file position to the start of the file and write something into the file. Then you close the file and flush the toilet, er, volume. This is the order of calls in our doSaveAs routine, the majority of which is error checking if anything goes wrong, goes wrong, goes wrong...

To write our graph out to disk, we simply have to write out our quickdraw picture. But first we write a 512 byte blank header. This will cause MacDraw to open the PICT file and use it's default settings. Our plotter program has no open function so we save our document as a MacDraw document so it can be viewed later and edited. The picture handle is stored in DrawingPic so we do a GetHandleSize and write out that number of bytes to the file, passing a pointer to the dereferenced picture handle. After saving the picture, we set the window title to the file name and store the file name, and volume reference number in our document data structure and enable the Save menu item for future use. For the Save function, we do the same thing as the SaveAs (they could be combined at the expense of clarity) except that we get the file name and volume reference number from our document datastructure rather than from a standard file dialog.

But Where Do We Actually Model It?

I told you it was tempting to get lost in the Macintosh interface and forget all about the modeling task our Boss assigned us to! We are supposed to be modeling a quadratic equation and coming up with some numbers for that next moon shot, not sitting around reading *Inside Macintosh* for the 100th time. Our Solve Unit implements the routines to analyze our equation and plot it

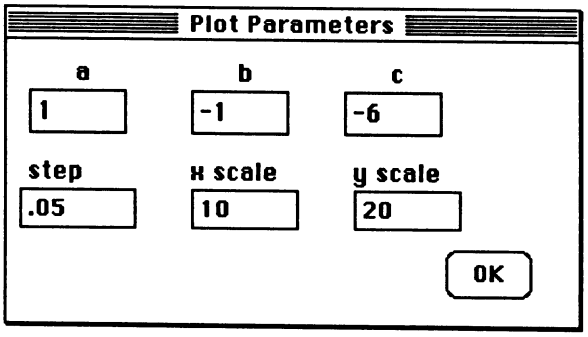


Fig. 4 Dialog Box gets our parameters

by actually creating the quickdraw picture we've been spending all this time talking about. We also put up a dialog box to get all of the equation parameters and plot characteristics we need. We can set the step size, and the scale of both the x axis and the y axis. And of course, we can input the three parameters that define the general quadratic equation. In our doDialog routine, we get these parameters from the user, and remember them so they appear the next time the plot function is selected. SetIText is used to set the dialog items with the values of the global variables that hold the equation parameters. A flag is used to by-pass SetIText if this is the first time we have been called, in which case the default values from the dialog item resources will be used. This allows the user to modify the default values in the resource file with ResEdit if he wishes.

The hard part about the dialog box is converting the user's response into real numbers. LS Pascal has a routine called ReadString that takes the string variables from the GetDItem trap and converts them into integer and real numbers for our equation parameters. However, this routine crashes mightily if the user types a non-numeric entry in the dialog! So the first improvement would be to protect yourself by checking the dialog input somehow.

The Quadratic Solution

The example problem is to find the roots of a quadratic equation. The problem in itself is not difficult (this time), but the solution technique will be similar for other problems.

First we should consider what the input specifications will be. Among the questions we should ask ourselves about the problem are:

1) What values will be provided as input to the program?
2) What range will these values be in?
3) What format will the values be in?
4) In what ways can these input values be used? Can the inputs be modified?
5) How will the input values be provided to the program?
6) What values will be returned by the program?
7) What format will the return values be in?
8) What kind of output will be produced?

Any quadratic equation may be reduced to the form, —

$ax^2 + bx + c = 0$

then

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

If a, b, and c are real then:
If b^2-4ac is positive, the roots are real and unequal;
If b^2-4ac is zero, the roots are real and equal;
If b^2-4ac is negative, the roots are imaginary and unequal.

The inputs required in this problem are a, b, and c. These values will be in the set of real numbers. Our dialog box allows us to input these values. On our plot, we print the results in a box indicating if the analysis yields real, or complex results. We can also solve the first derivative problem and determine where the graph has a local minimum or a maximum. Our program doesn't do it, but we could also indicate the second derivative to show if the curve is concave upward or downward. Keep in mind that in a "real life" problem the algorithm is not always given as in this example. The user normally doesn't care how the program arrives at the result as long as the result is correct. If too many details are specified early in the defining the problem, the programmer may get "locked in" to a particular solution too early. The programmer then lacks the flexibility needed to choose the best technique to solve the problem.

Our SolveIt and Quad routines implement the analysis portion of our modeling program by solving the quadratic equation and checking for the type of result. When the user selects Plot from the file menu, our doPlot routine will call SolveIt, which in turn calls Quad to get the "answers". With answers in hand, we call PlotMe to create a quickdraw picture and display it in our window. After creating the picture with OpenPicture and ClosePicture, we get the picture displayed by simply invalidating the window's portRect. This forces an update event, when will draw the graph by calling DrawPicture in our update event routine. Thus the whole plotter program really boils down to the magical routine which creates this quickdraw picture. This routine is called PrQDPicStuff in our Solve unit and is the most important of the whole program.

Creating the Quickdraw Picture

When we call OpenPicture, the characteristics of the current grafport will be used to determine how the picture will be stored. In our PrQDPicStuff routine, we want to allow for drawing either the picture on the screen or on the printer. We do this by passing to the routine, the display rectangle for the picture and a parameter indicating if we have been called from the Plot function or from the Print function. This was done to allow the added printer option of printing the graph using the full printer page, rather than the window's portRect. While printing to full page size is not WYSIWYG, it is useful to get the biggest plot possible when you are trying to analyze a function's behavior. In addition, if we are printing on the laserwriter, we set the font to times instead of geneva.

Constructing the picture definition routine is important. In our program we have grouped all the tasks together. First we calculate all the program variables, from the picture rectangle's width and height. We get the font information from GetFontInfo so our paragraph can be constructed properly. The initial x and y values are calculated. These real values are converted into integer coordinates for plotting in such a way that the plot axis will extend the full length and height of the picture rectangle.

Next all the text is constructed with LS Pascal's StringOf command and stored in Macintosh strings. For our paragraph, we use an array of strings which will be used to display the analysis results: the roots of the equation, the local min/max point, and the equation's defining parameters. Once all the initial calculations and strings are set up, we are ready to 'draw' into our picture.

We set the background, axis and graph colors as set by the color menu items checked. We draw a box around the graph boundary (the display rectangle). We draw the axis, the plot itself, and a box for our paragraph. Finally we draw the paragraph and dispose of our temporary handles and clip regions.

Pic Comments

In order to get MacDraw to display our graph, the objects we draw must be grouped together. In particular, the graph itself must be grouped so all those little line segments stick together. A Pic Comment is used to do this. Pic Comments allow commands to be imbedded within the picture definition. There are two published lists of Pic Comments in the technical notes. One is the Pic Comments that MacDraw recognizes. The other are the Pic Comments that Postscript recognizes. Using these two sets of Pic Comments will enhance an application's ability to create pictures that can be imported into other applications or that can take advantage of new capabilities. Apple is supposed to be keeping a list of registered Pic Comments, but little has come from it. The power of QuickDraw's Pic Comments has really not been exploited or pushed by Apple in comparison to postscript.

PostScript Pic Comments

Since Quickdraw cannot do hairlines, we need to use some postscript Pic Comments the LaserWriter understands to set the line width thinner than that possible by Quickdraw. To get a hairline, define:

```
Type
    widhdl = ^widptr;
    widptr=^widpt;
    widpt=Point;
Var
    Width:Widhdl;
```

Then set up Width as follows:

```
Width^^.h:=10;
Width^^.v:=1;
```

Then use a Pic Comment:

```
PicComment(SetLineWidth,2,Handle(Width));
```

We can also draw our text so that it will be grouped as a paragraph by using the TextBegin and TextEnd comments. With the PicGrpEnd and PicGrpBegin comments we can group objects so that they stay together when moved around in MacDraw.

Printing Our Graph

Once our picture is defined, we can print it when the user selects print from the file menu. To print, we call our PrQDStuff routine above with the printing rectangle obtained from the Print Setup dialog and the display device set to LaserWriter.

PrQDStuff will re-create the picture to the page rectangle, if the print option is selected, and a simple DrawPicture command during an open printing grafport will send the picture to the printer. The print traps are now in the system file, so we no longer link to the printing glue. In our doPrint routine, we get our print handle, saved in our document definition, save the current grafport and call PrOpen. Then we validate our print record and call PrJobDialog to put up the printing dialog box. If the user selects print, then we open the printing grafport by calling PrOpenDoc followed by PrOpenPage. From the printing info record (prInfo.rPage) we get the PageRect to pass to our picture creation routine. The PrintMe routine opens the picture and re-creates it using the PageRect, just as our PlotMe routine set up the picture for drawing to the screen. After drawing the picture into the printing grafPort, we close the page, close the document and close the printing manager by calling PrClosePage, PrCloseDoc and PrClose respectfully. When the printing grafPort is opened, the low level Quickdraw primitives are replaced with pointers into the printing manager, where the printer equivalent of the Quickdraw primitives translate the quickdraw picture definition into LaserPrep commands, which in turn are defined by the LaserPrep file into Postscript, and then interpreted by the Postscript interpreter in the LaserWriter into an image on the page. How does Postscript image the page? I don't know. If I did know, I'd build a Postscript photo-typesetter, buy an Adobe Atlas board and have myself a 2700 dot per inch typesetting machine without paying Allied Linotype $50,000.

New Goodies Not Addressed

The key to this program is the fact that it produces a document that can be read, edited and placed by two other popular programs. In this way, the Plotter Program illustrates an important rule in the MultiFinder world: applications should work together to produce a library of compatible tools. The best way this can happen is by formatting the output of similar tools to read and write compatible documents. The MacDraw PICT file, the MacPaint bit-map file and the MacWrite formatted text file are important document standards that many similar tools support, besides the nearly universal unformatted text file. Even though Apple has defined a standard style definition for TextEdit that allows for font changes, we have yet to see a widely used formatted text item type that is a standard across many applications.

Apple Share is another new goodie that affects developers. How do we make Apple Share compatible programs? Consider this: To be MultiFinder friendly, Apple says you should keep track of where your windows were moved last. That means you have to write something to disk to save this information, but where? If you write it in the resource fork, the most logical place, then your program can't be Apple Share compatible since the resource fork is not sharable. Two users cannot write to the same resource fork without destroying the integrity of the file. So Apple has created a catch-22 situation; supporting one new goodie (MultiFinder) makes it impossible to support another (Apple Share). Apple's recommended solution is to write a "configuration file" into the system folder. Great. Now our hard disks will have hundreds of little configuration files to keep track of! Since the resource manager is Apple Share brain damaged, it hardly seems realistic for Apple to expect developer's to rush madly to embrace this goodie. We encourage Apple to continue to define and support compatible data formats so tools can be used together.

```
{ Quadratic Equation Example program  }
{ By Dave Kelly & Dave Smith          }
{ ©MacTutor, 1988                     }

PROGRAM QuadraticPlotter;

   USES
      ROM85, PrintTraps, PlotGlobals, Misc, myPlotStuff;

   PROCEDURE crash;
   BEGIN
      ExitToShell;
   END;

   PROCEDURE InitMac;
      VAR
         err : OSErr;
   BEGIN
      MaxApplZone;
      MoreMasters;
      MoreMasters;
```

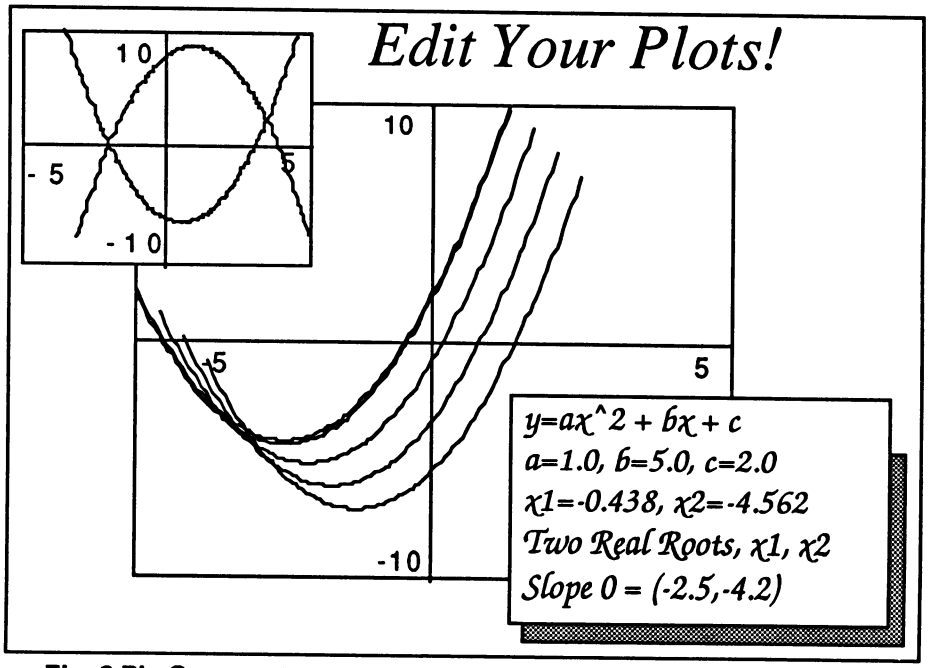


Fig. 6 Pic Comments group objects for editing in MacDraw. Note that MacDraw ignores hairline Pic Comment!

```
      MoreMasters;
      MoreMasters;
      MoreMasters;
      InitGraf(@thePort);
      InitFonts;
      InitWindows;
      InitMenus;
      TEInit;
      InitDialogs(@crash);
      InitCursor;
      FlushEvents(everyEvent, 0);
      Finished := false;
      dialogflg := false;

      color[1] := 33;   {black}
      color[2] := 30;   {white}
      color[3] := 205;  {red}
      color[4] := 341;  {green}
      color[5] := 409;  {blue}
      color[6] := 273;  {cyan}
      color[7] := 137;  {magenta}
      color[8] := 69;   {yellow}

      theWorld.machineType := 4;  {Mac II}
   {err := SysEnvirons(1, theWorld);  not in LSP 1.11}
      typeOfMac := theWorld.machineType;

      IF (typeOfMac >= 0) AND (NGetTrapAddress(WNETrapNum,
ToolTrap) = NGetTrapAddress(UnImplTrapNum, ToolTrap)) THEN
      BEGIN
          WNE := FALSE;
          doMessage('WaitNextEvent not implemented', '', '',
''); {Only true under Multifinder}
      END
      ELSE
          WNE := TRUE;
      mouseRgn := NewRgn;
   END;

   PROCEDURE initRects;
      VAR
          MemoryPtr : ^Integer;
   BEGIN
      MemoryPtr := pointer(mBarHeightGlobal);
      mBarHeight := MemoryPtr^;
      screen := ScreenBits.Bounds; {current screen device}
      SetRect(DragArea, Screen.left + 4, Screen.top + mBar-
Height + 4, Screen.right - 4, Screen.bottom - 4);
      SetRect(GrowArea, Screen.left + MinWidth, Screen.top +
MinHeight, Screen.right - 8, Screen.bottom - 8);
      SetRect(PlotWindowRect, Screen.left + 15, Screen.top +
45, Screen.right - 75, Screen.bottom - 50);
      SetRect(ZoomRect, Screen.left + 4, Screen.top + mBar-
Height + 4, Screen.right - 4, Screen.bottom - 4);
   END;

   PROCEDURE InitMyWindow;
      VAR
          windtype : integer;
          Visible : boolean;
          GoAway : boolean;
          RefVal : LongInt;
          title : str255;
          myPrint : THPrint;
   BEGIN
      Visible := false;
      windtype := documentProc + ZoomBox;
      GoAway := true;
      RefVal := 0;
      title := 'Quadratic Plotter';
      PlotWindow := NewWindow(NIL, PlotWindowRect, title,
Visible, windtype, pointer(-1), GoAway, RefVal);
      PlotWindowPeek := WindowPeek(PlotWindow);
      SetPort(PlotWindow);
      TextFont(Geneva);
      TextSize(10);
      TextFace([]);  {plain}
      TextMode(1);   {Or}
      PenNormal;
      ForeColor(blackColor);
      BackColor(whiteColor);
      PlotDocHandle := DocHandle(NewHandle(sizeof(Document)));
      PlotWindowPeek^.refCon := LongInt(PlotDocHandle);
      PlotWindowPeek^.windowKind := userKind;
      PlotDocHandle^^.drawing := NIL;
      WITH PlotWindow^.portRect DO
          BEGIN
          SetRect(VCRect, right - (SBarWidth - 1), top - 1,
right + 1, bottom - (SBarWidth - 2));
          SetRect(HCRect, left - 1, bottom - (SBarWidth - 1),
right - (SBarWidth - 2), bottom + 1);
          SetRect(GrowRect, HCRect.right, HCRect.top,
VCRect.right, HCRect.bottom);
          SetRect(PicRect, left, top, right - (SBarWidth -
1), bottom - (SBarWidth - 1));
          SetRect(PageRect, left, top, right - (SBarWidth -
1), bottom - (SBarWidth - 1));
          END;  {of with }
      ClipRect(PlotWindow^.portRect);
      myPrint := THPrint(NewHandle(SIZEOF(TPrint)));
      PlotDocHandle^^.Print := myPrint;
   END;

   PROCEDURE InitMyMenus;
      VAR
          i : integer;
   BEGIN
      myMenus[AppleM] := GetMenu(AppleMenu);
      AddResMenu(myMenus[AppleM], 'DRVR');
      myMenus[FileM] := GetMenu(FileMenu);
      myMenus[EditM] := GetMenu(EditMenu);
      myMenus[ColorM] := GetMenu(ColorMenu);
      myMenus[OptionM] := GetMenu(OptionMenu);
      myMenus[GraphM] := GetMenu(GraphMenu);
      myMenus[AxisM] := GetMenu(AxisMenu);
      myMenus[BackgroundM] := GetMenu(BackgroundMenu);

      FOR i := 1 TO MenuCount DO
          InsertMenu(myMenus[i], 0);
      FOR i := SubMenuStart TO TotalMenuCount DO
          InsertMenu(myMenus[i], -1);

      GraphColor := 3;
      CheckItem(myMenus[GraphM], GraphColor, true);
      AxisColor := 1;
      CheckItem(myMenus[AxisM], AxisColor, true);
      BackgroundColor := 2;
      CheckItem(myMenus[BackgroundM], BackgroundColor, true);

      CheckItem(myMenus[OptionM], 1, true);
      Option := 1;  {window rect printing}

      DrawMenuBar;
   END;   {of proc}
```


© The Definitive MacTutor, Vol. 4

```
    PROCEDURE doMouse (myEvent : EventRecord);
       VAR
            whereIsIt : integer;
            whichWindow : WindowPtr;
            localPt, globalPt : Point;
            oldPort : GrafPtr;
    BEGIN
       globalPt := myEvent.where;
       localPt := globalPt;     {global coord of mouse}
       GlobalToLocal(localPt);  {local coord of mouse}
       whereIsIt := FindWindow(globalPt, whichWindow);
       CASE whereIsIt OF
            inDesk :          {0}
               doMessage('Mouse Click on Desktop.', '(Not
handled in this program.)', '', '');
            inMenuBar :    {1}
               doMenuBar(MenuSelect(globalPt));
            inSysWindow : {2}
               SystemClick(myEvent, whichWindow);
            inContent :       {3}
               doContent(myEvent, whichWindow);
            inDrag :          {4}
               doDrag(whichWindow, globalPt);
            inGrow :       {5}
               doGrow(whichWindow, globalPt, False);
            inGoAway :     {6}
               IF TrackGoAway(whichWindow, globalPt) THEN
                  HideWindow(whichWindow);
            inZoomIn, InZoomOut :       {7, 8}
            BEGIN
            IF TrackBox(whichWindow, globalPt, whereIsIt) THEN
                  BEGIN
                     GetPort(OldPort);
                     SetPort(whichWindow); {safety device}
                     EraseRect(whichWindow^.portRect);
                     ZoomWindow(whichWindow, whereIsIt, True);
                     doGrow(whichWindow, globalPt, True);
                     SetPort(OldPort);
                  END;
            END;
            OTHERWISE
               BEGIN
               END;
       END; {of whereIsIt}
    END;

    PROCEDURE doKeyDowns (myEvent : EventRecord);
       VAR
            ch : char;
            charCode : longInt;
            keyCode : longInt;
    BEGIN
       charCode := BitAnd(myEvent.Message, charCodeMask);
keyCode := BitShift(BitAnd(myEvent.Message, keyCodeMask), -8);
       ch := Chr(charCode);  {get keyboard char}
       IF BitAnd(myEvent.Modifiers, CmdKey) = CmdKey THEN
            doMenuBar(MenuKey(ch))  { do menu command key}
       ELSE
            BEGIN  { do keystroke }
               ParamText('No typing supported…', '', '', '');
               itemhit := CautionAlert(AlertDialog, NIL);
            END;  { of do key stroke }
    END;  { of proc}

    PROCEDURE doUpdates (myEvent : EventRecord);
       VAR
            UpdateWindow : WindowPtr;
            TempPort : GrafPtr;
    BEGIN
       UpdateWindow := WindowPtr(myEvent.message);
       IF UpdateWindow = PlotWindow THEN
            BEGIN
               GetPort(TempPort); {save port}
               SetPort(UpdateWindow);
               BeginUpDate(UpdateWindow);
               BackColor(whiteColor);
               EraseRect(UpdateWindow^.portRect);
               ClipRect(UpdateWindow^.portRect);
               DrawPicture(DrawingPic, PicRect);
               DrawGrowIcon(UpdateWindow);
               EndUpDate(UpdateWindow);
               SetPort(TempPort);  {restore port}
            END;
    END; {of proc}

    PROCEDURE doActivates (myEvent : EventRecord);
       VAR
            TargetWindow : WindowPtr;
            TargetPeek : WindowPeek;
    BEGIN
       TargetWindow := WindowPtr(myEvent.message);
       TargetPeek := windowPeek(TargetWindow);
       SetPort(TargetWindow);
       IF Odd(myEvent.modifiers) THEN
            BEGIN {activate}
               IF TargetWindow = PlotWindow THEN
                  BEGIN
                     DisableItem(myMenus[EditM], eUndo);
                     DisableItem(myMenus[EditM], eCut);
                     DisableItem(myMenus[EditM], eCopy);
                     DisableItem(myMenus[EditM], ePaste);
                     DisableItem(myMenus[EditM], eClear);
                     DrawGrowIcon(TargetWindow);
                  END
            END  { of activate loop}
       ELSE
            BEGIN {deactivate}
               IF TargetWindow = PlotWindow THEN
                  BEGIN
                     EnableItem(myMenus[EditM], eUndo);
                     EnableItem(myMenus[EditM], eCut);
                     EnableItem(myMenus[EditM], eCopy);
                     EnableItem(myMenus[EditM], ePaste);
                     EnableItem(myMenus[EditM], eClear);
                     DrawGrowIcon(TargetWindow);
                  END; { of my window activation}
            END; {of deactivate loop}
    END; {of proc}

    PROCEDURE doMulti (myEvent : EventRecord);

       CONST
            MouseMovedEvt = $FA;
       VAR
            HiByte : byte;
            bit0 : LongInt;
            sysresult : boolean;
            ResumeWindow : WindowPtr;
            ResumePeek : WindowPeek;
            SuspendWindow : WindowPtr;
            SuspendPeek : WindowPeek;
            MouseMove : LongAndByte;
    BEGIN
```

```
    bit0 := 31; {convert 68000 to toolbox}
    MouseMove.longView := myEvent.message;
    HiByte := Byte(MouseMove.byteView.byte0);
    IF HiByte = MouseMovedEvt THEN
        BEGIN {Handle mouse moved event}
        END;
    {check for resume event }
    IF Odd(myEvent.message) THEN
        BEGIN {resume}
        {activate Event}
           ResumeWindow := FrontWindow;
           IF ResumeWindow = PlotWindow THEN
              BEGIN
                 SetPort(ResumeWindow);
                 InvalRect(ResumeWindow^.portRect);
                 DisableItem(myMenus[EditM], eUndo);
                 DisableItem(myMenus[EditM], eCut);
                 DisableItem(myMenus[EditM], eCopy);
                 DisableItem(myMenus[EditM], ePaste);
                 DisableItem(myMenus[EditM], eClear);
                 DrawGrowIcon(ResumeWindow);
              END;
           IF FrontWindow <> NIL THEN
              BEGIN {DA check}
                 ResumePeek := WindowPeek(FrontWindow);
                 IF ResumePeek^.windowKind < 0 THEN {DA}
                    BEGIN
                       myEvent.what := activateEvt;
                       BitSet(@myEvent.modifiers, bit0);
                       sysresult := SystemEvent(myEvent);
                    END; {da}
              END; {DA check}
    { end of activate Event}
        END {of resume}
    ELSE
        BEGIN {suspend}
    {de-activate Event}
           SuspendWindow := FrontWindow;
           IF SuspendWindow = PlotWindow THEN
              BEGIN {plotwindow}
                 SetPort(SuspendWindow);
                 InvalRect(SuspendWindow^.portRect);
                 EnableItem(myMenus[EditM], eUndo);
                 EnableItem(myMenus[EditM], eCut);
                 EnableItem(myMenus[EditM], eCopy);
                 EnableItem(myMenus[EditM], ePaste);
                 EnableItem(myMenus[EditM], eClear);
                 DrawGrowIcon(SuspendWindow);
              END; {plotwindow}
           IF FrontWindow <> NIL THEN
              BEGIN {DA check}
                 SuspendPeek := WindowPeek(FrontWindow);
                 IF SuspendPeek^.windowKind < 0 THEN
                    BEGIN {da}
                       myEvent.what := activateEvt;
                       BitClr(@myEvent.modifiers, bit0);
                       sysresult := SystemEvent(myEvent);
                    END; {da}
              END; {DA check}
    { end of de-activate Event}
        END; {suspend}
END; {of proc}

PROCEDURE MainEventLoop;
    CONST
        MultiFinderEvt = 15;
    VAR
        sleep : LongInt;
        Event : EventRecord;
        DoIt : Boolean;
BEGIN
    sleep := 10;
    REPEAT
    IF WNE THEN
    DoIt := WaitNextEvent(EveryEvent, Event, sleep, NIL)
        ELSE
           BEGIN
              SystemTask;
              DoIt := GetNextEvent(EveryEvent, Event);
           END;
        IF DoIt THEN
           CASE Event.what OF
              mouseDown :
                 doMouse(Event);
              KeyDown, Autokey :
                 doKeyDowns(Event);
              updateEvt :
                 doUpdates(Event);
              activateEvt :
                 doActivates(Event);
              MultiFinderEvt :
                 doMulti(Event);
              OTHERWISE
                 BEGIN
                 END;
           END;  {of event case}
    UNTIL Finished;  {end program}
END;

{ Main Program}
BEGIN
   InitMac;
   InitRects;
   InitMyWindow;
   InitMyMenus;
   MainEventLoop;
END.


UNIT MyPlotStuff;

INTERFACE

   USES
      ROM85, PrintTraps, PlotGlobals, Misc, solve, MyFileS-
tuff, MyPrintStuff;

   PROCEDURE doAbout;
   PROCEDURE doQuit;
   PROCEDURE doMenubar (menuResult : LongInt);
   PROCEDURE doContent (ConEvent : EventRecord;
                        contentWindow : windowPtr);
   PROCEDURE doDrag (GrabWindow : WindowPtr;
                     GlobalMouse : point);
   PROCEDURE doGrow (ResizeWindow : WindowPtr;
                     Globalmouse : point;
                     Zoomflg : Boolean);

IMPLEMENTATION

   PROCEDURE doAbout;
      VAR
         IDStrHandle : StringHandle;
         dialogP : DialogPtr;
         item : integer;
```

```
        Str1, Str2, Str3 : str255;
        myHeapSpace : LongInt;
        FreeSpace : Size;
BEGIN
    IDStrHandle := StringHandle(GetResource(rsrc, 0));
    IF IDStrHandle = NIL THEN
        BEGIN
            doMessage('Get About box crash!', '', '', '');
            ExitToShell;
        END;
    MoveHHi(Handle(IDStrHandle));
    HLock(Handle(IDStrHandle));
    FreeSpace := FreeMem;
    myHeapSpace := MaxMem(FreeSpace);
    NumToString(myHeapSpace, Str2);
    Str2 := concat('Memory = ', Str2);
    Str3 := '';
    Str1 := '';
    ParamText(IDStrHandle^^, Str1, Str2, Str3);
    dialogP := GetNewDialog(AboutDialog, NIL, pointer(-1));
    IF dialogP = NIL THEN
    BEGIN
    doMessage('Dialog crash!', 'We are dead...', '', '');
    ExitToShell;
    END;
    initCursor;
    ModalDialog(NIL, item);
    DisposDialog(dialogP);
    HUnlock(Handle(IDStrHandle));
END;

PROCEDURE doQuit;
BEGIN
    BackColor(whiteColor);
    ForeColor(blackColor);
    PenNormal;
    DisposeWindow(PlotWindow);
    DisposeRgn(mouseRgn);
    DisposHandle(Handle(PlotDocHandle));
    Finished := true;
END; (of proc)

PROCEDURE doMenubar; (menuResult : LongInt)
    VAR
        theMenu : integer;
        theItem : integer;
        daName : STR255;
        accItem : integer;
        temp : GrafPtr;
BEGIN
    theMenu := HiWord(menuResult); (menu)
    theItem := LoWord(menuResult); (item)
    CASE theMenu OF
    AppleMenu :
        BEGIN
            IF theItem = aAbout THEN
                doAbout
            ELSE
                BEGIN (must be DA)
                    GetItem(myMenus[AppleM], theItem, daName);
                    GetPort(temp);   (protect against flacky DA)
                    accItem := OpenDeskAcc(daName);
                    SetPort(temp);
                END; (else)
        END; (of AppleMenu)
    FileMenu :
                BEGIN
                    CASE theItem OF
                        fPlot :
                            BEGIN
                                doPlot;
                            END;
                        fSave :
                            BEGIN
                                doSave;
                            END;
                        fSaveAs :
                            BEGIN
                                doSaveAs;
                            END;
                        fPageSet :
                            BEGIN
                                doPageSet;
                            END;
                        fPrint :
                            BEGIN
                                doPrint;
                            END;
                        fQuit :
                            BEGIN
                                doQuit;
                            END;
                        OTHERWISE
                            BEGIN
                            END;
                    END; (of theitem)
                END; (of FileMenu)
    EditMenu :
        BEGIN
            IF NOT SystemEdit(theitem - 1) THEN
                BEGIN
                    CASE theItem OF
                        eUndo :
                            BEGIN
                            END;
                                eCut :
                                    BEGIN
                                    END;
                                eCopy :
                                    BEGIN
                                    END;
                                ePaste :
                                    BEGIN
                                    END;
                                eClear :
                                    BEGIN
                                    END;
                                OTHERWISE
                                    BEGIN
                                    END;
                            END; (of case)
                        END; (of system edit)
            END; (of EditMenu)
        ColorMenu :
            BEGIN (just a dummy for submenus)
            END; (of color menu)
        GraphMenu :
            BEGIN
            CheckItem(myMenus[GraphM], GraphColor, false);
            GraphColor := theitem;
            CheckItem(myMenus[GraphM], GraphColor, true);
            END; (of graph menu)
        AxisMenu :
            BEGIN
```

```
            CheckItem(myMenus[AxisM], AxisColor, false);
            AxisColor := theitem;
            CheckItem(myMenus[AxisM], AxisColor, true);
         END; (of axis menu)
      BackgroundMenu :
      BEGIN
   CheckItem(myMenus[BackgroundM], BackgroundColor, false);
      BackgroundColor := theitem;
   CheckItem(myMenus[BackgroundM], BackgroundColor, true);
         END; (of background menu)
      OptionMenu :
         BEGIN
            CheckItem(myMenus[OptionM], Option, false);
            Option := theitem;
            CheckItem(myMenus[OptionM], Option, true);
         END;
      OTHERWISE
         BEGIN
         END;
   END; (of theMenu)
   HiliteMenu(0); (un-hilite selected menu)
END;

PROCEDURE doContent; ((ConEvent : EventRecord)
                        (contentWindow : windowPtr);)
   VAR
      localPt, globalPt : Point;
      part : integer;
      myRect : Rect;
      control : ControlHandle;
BEGIN
   IF contentWindow <> FrontWindow THEN
      SelectWindow(contentWindow);
   globalPt := ConEvent.where;
   localPt := globalPt;     (global coord of mouse)
   GlobalToLocal(localPt); (local coord of mouse)
   part := FindControl(localPt, contentWindow, control);

   IF contentWindow = PlotWindow THEN
      BEGIN
         SetPort(PlotWindow);
         IF part <> 0 THEN
            BEGIN (in control)
            END;
         IF part = 0 THEN
            BEGIN (content region)
               myRect := PlotWindow^.portRect;
               IF PtInRect(localPt, myRect) THEN
                  BEGIN
                  END; (of ptInRect)
            END; ( of part=0 )
      END; (of contentwindow)
END;  (of proc)

PROCEDURE doDrag; ((GrabWindow : WindowPtr)
                     (GlobalMouse : point);)
BEGIN
   DragWindow(GrabWindow, GlobalMouse, DragArea);
END;

PROCEDURE doGrow; ((ResizeWindow : WindowPtr;)
                     (Globalmouse : point;)
                     (ZoomFlg:Boolean);)
   VAR
      newSize : LongInt;
      hsize : integer;
      vsize : integer;
      oldPort : GrafPtr;
      myRect : rect;
      tempLong : LongInt;
BEGIN
   IF (ResizeWindow <> FrontWindow) THEN
      SelectWindow(ResizeWindow)
   ELSE
      BEGIN
         IF (ZoomFlg) THEN
            BEGIN
               WITH ResizeWindow^.portRect DO
                  BEGIN
                  tempLong := bottom - top;
                  newSize := BitShift(tempLong, 16);
                  newSize := newSize + (right - left);
                  END;
            END
         ELSE
newSize := GrowWindow(ResizeWindow, Globalmouse, GrowArea);
         IF newSize <> 0 THEN
         BEGIN  (grow the window)
         hsize := LoWord(newSize);
         vsize := HiWord(newSize);
         IF ResizeWindow = PlotWindow THEN
            BEGIN
            WITH ResizeWindow^.portRect DO (Pre-Grow)
            BEGIN
SetRect(VCRect, right - (SBarWidth - 1), top - 1, right + 1,
bottom - (SBarWidth - 2));
SetRect(HCRect, left - 1, bottom - (SBarWidth - 1), right -
(SBarWidth - 2), bottom + 1);
SetRect(GrowRect, HCRect.right, HCRect.top, VCRect.right,
HCRect.bottom);
SetRect(PicRect, left, top, right - (SBarWidth - 1), bottom -
(SBarWidth - 1));
            END;  (of with )
            InvalRect(VCRect);
            InvalRect(HCRect);
            InvalRect(GrowRect);
SizeWindow(ResizeWindow, hsize, vsize, TRUE); (new portRect)
            WITH ResizeWindow^.portRect DO (Post Grow)
            BEGIN
SetRect(VCRect, right - (SBarWidth - 1), top - 1, right + 1,
bottom - (SBarWidth - 2));
SetRect(HCRect, left - 1, bottom - (SBarWidth - 1), right -
(SBarWidth - 2), bottom + 1);
SetRect(GrowRect, HCRect.right, HCRect.top, VCRect.right,
HCRect.bottom);
SetRect(PicRect, left, top, right - (SBarWidth - 1), bottom -
(SBarWidth - 1));
SetRect(PageRect, left, top, right - (SBarWidth - 1), bottom -
(SBarWidth - 1));
            END;  (of with )
            InvalRect(VCRect);
            InvalRect(HCRect);
            InvalRect(GrowRect);
            InvalRect(PicRect);
         END; (of if ResizeWindow)
      END; (of grow window stuff)
   END; (of if then newsize)
END;  ( of proc )

END. (of unit)


UNIT MyFileStuff;
```


© The Definitive MacTutor, Vol. 4

```
INTERFACE

   USES
      ROM85, PrintTraps, PlotGlobals, Misc;

   PROCEDURE doSaveAs;
   PROCEDURE doSave;

IMPLEMENTATION

   PROCEDURE doSaveAs;
      LABEL
          1, 2;
      CONST
          SFPutLeft = 82;
          SFPutTop = 50;
          headerBytes = 512;
      TYPE
          DrawHeader = RECORD
                fill : ARRAY[1..256] OF integer;
             END;
      VAR
          SFPutPt : Point;
          theReply : SFReply;
          err : OSErr;
          refNum : Integer;
          bytes : LongInt;
          myWindow : WindowPtr;
          title : str255;
          myType : OSType;
          myCreator : OSType;
          str1, str2 : str255;
          header : DrawHeader;
          i : integer;
          myPicture : PicHandle;
          PicLength : LongInt;
   BEGIN
      myPicture := PlotDocHandle^^.Drawing;
      IF myPicture = NIL THEN
          doMessage('No picture to save yet!', '', '', '')
      ELSE
          BEGIN
             SetPt(SFPutPt, SFPutLeft, SFPutTop);
             WITH header DO
                BEGIN
                FOR i := 1 TO headerBytes DIV 2 DO
                   fill[i] := 0;
                END;
             bytes := headerBytes;
             myWindow := PlotWindow;
             GetWTitle(myWindow, title);
   SFPutFile(SFPutPt, 'Save Plot as…', title, NIL, theReply);
             IF theReply.good THEN
                BEGIN {theReply.good}
                myType := 'PICT';
                myCreator := 'MDRW'; {MacDraw doc}
   err := Create(theReply.fname, theReply.vRefNum, myCreator,
myType);
                IF err <> noErr THEN
                BEGIN {err<>noErr}
                   IF err = dupFNErr THEN
                   BEGIN {err=dupFNerr}
   err := FSDelete(theReply.fname, theReply.vRefNum);
                   IF err <> noErr THEN
                   BEGIN {err<>noErr}
   doMessage('Cannot delete duplicate file.', '', '', '');
                   GOTO 1;
                   END; {err<>noErr}
   err := Create(theReply.fname, theReply.vRefNum, myCreator,
myType);
                   IF err <> noErr THEN
                   BEGIN {err<>noErr}
   doMessage('Cannot create file...', '', '', '');
                      GOTO 1;
                   END; {err<>noErr}
                   END {err=dupFNerr}
                   ELSE {create file error}
                      BEGIN {create file error}
   doMessage('Cannot create new file...', '', '', '');
                      GOTO 1;
                      END; {create file error}
                END; {err<>noErr}
   err := FSOpen(theReply.fname, theReply.vRefNum, refNum);
                IF err <> noErr THEN
                   BEGIN
   doMessage('Cannot open file...', '', '', '');
                   GOTO 1;
                   END;
                err := SetFPos(refNum, FSFromStart, 0);
                IF err <> noErr THEN
                   BEGIN
   doMessage('Cannot set start of file...', '', '', '');
                   GOTO 2;
                   END;

                err := FSWrite(refNum, bytes, @header);
                IF err <> noErr THEN
                   BEGIN
   doMessage('Cannot write header to file...', '', '', '');
                   GOTO 2;
                   END;
                IF bytes <> 512 THEN
                   BEGIN
                      NumToString(bytes, str1);
                      str1 := concat(str1, ' bytes');
                      str2 := concat('out of ', '512');
   doMessage('Only able to write ', str1, str2, 'to file.');
                      GOTO 2;
                   END;
   PicLength := GetHandleSize(Handle(DrawingPic));
                bytes := PicLength;
   err := FSWrite(refNum, bytes, pointer(DrawingPic^));
                IF err <> noErr THEN
                   BEGIN
   doMessage('Cannot write picture to file...', '', '', '');
                   GOTO 2;
                   END;
                IF bytes <> PicLength THEN
                   BEGIN
                      NumToString(bytes, str1);
                      str1 := concat(str1, ' bytes');
                      NumToString(PicLength, str2);
                      str2 := concat('out of ', str2);
   doMessage('Only able to write ', str1, str2, 'to file.');
                      GOTO 2;
                   END;
                SetWTitle(myWindow, theReply.fname);
                PlotDocHandle^^.FileName := theReply.fname;
   PlotDocHandle^^.VolRefNum := theReply.vRefNum;
                EnableItem(myMenus[FileM], fSave);
2 :
                err := FSClose(refNum);
                IF err <> noErr THEN
                   BEGIN
```

```
    doMessage('Cannot close file...', '', '', '');
                    ExitToShell;
                    END;
                err := FlushVol(NIL, theReply.vRefNum);
                IF err <> NoErr THEN
                    BEGIN
            doMessage('Cannot flush volume...', '', '', '');
                    ExitToShell;
                    END;
1 :
                SetCursor(arrow);
            END; {if good}
        END; {else pic exits}
    END;{ of proc}

    PROCEDURE doSave;
        LABEL
            1, 2;
        CONST
            headerBytes = 512;
        TYPE
            DrawHeader = RECORD
                    fill : ARRAY[1..256] OF integer;
                END;
        VAR
            err : OSErr;
            refNum : Integer;
            bytes : LongInt;
            myWindow : WindowPtr;
            title : str255;
            str1, str2 : str255;
            header : DrawHeader;
            i : integer;
            myPicture : PicHandle;
            PicLength : LongInt;
            myRefNum : integer;
            myFname : str255;
    BEGIN
        myPicture := PlotDocHandle^^.Drawing;
        IF myPicture = NIL THEN
            doMessage('No picture to save yet!', '', '', '')
        ELSE
            BEGIN
                myRefNum := PlotDocHandle^^.VolRefNum;
                myFname := PlotDocHandle^^.FileName;
                IF myRefNum = 0 THEN
                BEGIN
        doMessage('Cannot save file', 'Use SaveAs...', '', '');
                GOTO 1;
                END;

                WITH header DO
                    BEGIN
                    FOR i := 1 TO headerBytes DIV 2 DO
                        fill[i] := 0;
                    END;
                bytes := headerBytes;

                err := FSOpen(myFname, myRefNum, refNum);
                IF err <> noErr THEN
                    BEGIN
                doMessage('Cannot open file...', '', '', '');
                    GOTO 1;
                    END;
                err := SetFPos(refNum, FSFromStart, 0);
                IF err <> noErr THEN
                    BEGIN
        doMessage('Cannot set start of file...', '', '', '');
                        GOTO 2;
                    END;

                err := FSWrite(refNum, bytes, @header);
                IF err <> noErr THEN
                    BEGIN
        doMessage('Cannot write header to file...', '', '', '');
                        GOTO 2;
                    END;
                IF bytes <> 512 THEN
                    BEGIN
                        NumToString(bytes, str1);
                        str1 := concat(str1, ' bytes');
                        str2 := concat('out of ', '512');
    doMessage('Only able to write ', str1, str2, 'to file.');
                        GOTO 2;
                    END;
                PicLength := GetHandleSize(Handle(DrawingPic));
                bytes := PicLength;
    err := FSWrite(refNum, bytes, pointer(DrawingPic^));
                IF err <> noErr THEN
                    BEGIN
    doMessage('Cannot write picture to file...', '', '', '');
                        GOTO 2;
                    END;
                IF bytes <> PicLength THEN
                    BEGIN
                        NumToString(bytes, str1);
                        str1 := concat(str1, ' bytes');
                        NumToString(PicLength, str2);
                        str2 := concat('out of ', str2);
    doMessage('Only able to write ', str1, str2, 'to file.');
                        GOTO 2;
                    END;
2 :
                err := FSClose(refNum);
                IF err <> noErr THEN
                    BEGIN
                doMessage('Cannot close file...', '', '', '');
                    ExitToShell;
                    END;
                err := FlushVol(NIL, myRefNum);
                IF err <> NoErr THEN
                    BEGIN
                doMessage('Cannot flush volume...', '', '', '');
                    ExitToShell;
                    END;
1 :
                SetCursor(arrow);
            END; {if good}
    END;{ of proc}

END.


UNIT solve;

INTERFACE

    USES
        ROM85, PrintTraps, PlotGlobals, Misc;

    PROCEDURE quad (a, b, c : real;
                                VAR x1, x2 : real;
                                VAR result : integer);
```

```
    FUNCTION solveit : integer;
    PROCEDURE doPlot;
    PROCEDURE PrQDStuff (pRect : rect;
                         QDdevice : integer);

IMPLEMENTATION

    FUNCTION positivecalc (a, b, check : real) : real;
    BEGIN
       positivecalc := (-b + sqrt(check)) / (2 * a);
    END;

    FUNCTION negativecalc (a, b, check : real) : real;
    BEGIN
       negativecalc := (-b - sqrt(check)) / (2 * a);
    END;

    PROCEDURE doDialog;
       VAR
           dialogP : DialogPtr;
           item : integer;
           dtype : integer;
           ditem : handle;
           drect : rect;
           dtext : Str255;
    BEGIN
       dialogP := GetNewDialog(ParamDialog, NIL, pointer(-1));
       IF dialogP = NIL THEN
           BEGIN
       doMessage('Dialog crash!', 'We are dead...', '', '');
           ExitToShell;
           END;
       initCursor;
       IF dialogflg THEN
           BEGIN
              dtext := StringOf(a : 4 : 1);
              GetDItem(dialogP, dA, dtype, ditem, drect);
              SetIText(ditem, dtext);
              dtext := StringOf(b : 4 : 1);
              GetDItem(dialogP, dB, dtype, ditem, drect);
              SetIText(ditem, dtext);
              dtext := StringOf(c : 4 : 1);
              GetDItem(dialogP, dC, dtype, ditem, drect);
              SetIText(ditem, dtext);
              dtext := StringOf(step : 5 : 3);
              GetDItem(dialogP, dSTEP, dtype, ditem, drect);
              SetIText(ditem, dtext);
              dtext := StringOf(xscale);
              GetDItem(dialogP, dXSCALE, dtype, ditem, drect);
              SetIText(ditem, dtext);
              dtext := StringOf(yscale);
              GetDItem(dialogP, dYSCALE, dtype, ditem, drect);
              SetIText(ditem, dtext);
           END;
       REPEAT
           ModalDialog(NIL, item);
       UNTIL item = dOK;
       GetDItem(dialogP, dA, dtype, ditem, drect);
       GetIText(ditem, dtext);
       ReadString(dtext, a);
       GetDItem(dialogP, dB, dtype, ditem, drect);
       GetIText(ditem, dtext);
       ReadString(dtext, b);
       GetDItem(dialogP, dC, dtype, ditem, drect);
       GetIText(ditem, dtext);
       ReadString(dtext, c);
       GetDItem(dialogP, dSTEP, dtype, ditem, drect);
       GetIText(ditem, dtext);
       ReadString(dtext, step);
       GetDItem(dialogP, dXSCALE, dtype, ditem, drect);
       GetIText(ditem, dtext);
       ReadString(dtext, xscale);
       GetDItem(dialogP, dYSCALE, dtype, ditem, drect);
       GetIText(ditem, dtext);
       ReadString(dtext, yscale);
       PlotDocHandle^^.aParam := a;
       PlotDocHandle^^.bParam := b;
       PlotDocHandle^^.cParam := c;
       PlotDocHandle^^.stepParam := step;
       PlotDocHandle^^.xParam := xscale;
       PlotDocHandle^^.yParam := yscale;
       dialogflg := true;
       DisposDialog(dialogP);
    END;

    PROCEDURE quad; {(a, b, c : real;var x1, x2 : real;var
result : integer);}
       VAR
           check : real;
    BEGIN
       result := 0;
       check := (b * b) - (4 * a * c);
       IF result = 0 THEN
           BEGIN
              { Check if double root exists }
              IF check = 0 THEN
                 BEGIN
                    result := 2;
                    x1 := positivecalc(a, b, check);
                    x2 := x1;
                 END;
              { Check if real result}
              IF check > 0 THEN
                 BEGIN
                    result := 1;
                    x1 := positivecalc(a, b, check);
                    x2 := negativecalc(a, b, check);
                 END;
              { Check if root is complex }
              IF check < 0 THEN
                 BEGIN
                    result := 3;
                    check := -check;
                    x1 := positivecalc(a, b, check);
                    x2 := negativecalc(a, b, check);
                 END;
           END;
    END;

    PROCEDURE PrQDStuff; {(pRect : rect; QDdevice : integer);}
       CONST
           Display = 1;
           LaserWriter = 2;
           ImageWriter = 3;
           NoJust = 0;
           LeftJust = 1;
           CenterJust = 2;
           RightJust = 3;
           FullJust = 4;
           LinesInParagraph = 5;
           {selected MacDraw comments}
           picDwgBeg = 130;
           picDwgEnd = 131;
           picGrpBeg = 140;
```

```
        picGrpEnd = 141;
        TextBegin = 150;
        TextEnd = 151;
        StringBegin = 151;
        StringEnd = 153;
        TextCenter = 154;
        {postscript comments}
        SetLineWidth = 182;
        PostScriptBegin = 190;
        TextIsPostscript = 194;
        PostScriptEnd = 191;
    TYPE
        widhdl = ^widptr;
        widptr = ^widpt;
        widpt = Point;

        TTxtPicRec = PACKED RECORD
              tJus : Byte;
              tFlip : Byte;
              tRot : Integer;
              tLine : Byte;
              tCmnt : Byte;
           END;

    VAR
        le, tp, ri, bo : integer;
        str1, str2, str3, str4, str5 : str255;
        str6, str7, str8, str9 : str255;
        hPos, vPos, hor, ver : integer;
        x, y, z1, z2 : real;
        rBox, ClipBox : rect;
        Width : Widhdl;
        leading : integer;
        LineNo : integer;
        ParagraphBegin : Point;
        Indent : integer;
        Paragraph : ARRAY[1..LinesInParagraph] OF str255;
        TxtPicRec : TTxtPicRec;
        TxtPicPtr : QDPtr;
        TxtPicHdl : QDHandle;
        TextClipRgn : RgnHandle;
        SaveClip : RgnHandle;
        fInfo : FontInfo;
BEGIN
    SaveClip := NewRgn;
    GetClip(SaveClip);
    ClipRect(pRect);
    TextClipRgn := NewRgn;
    penNormal;
    IF QDdevice = LaserWriter THEN
        BEGIN
           TextFont(times);
           TextSize(10);
           TextFace([]);
           TextMode(srcOr);
        END;
    hor := (pRect.right - pRect.left) DIV 2;
    ver := (pRect.bottom - pRect.top) DIV 2;
    Width := Widhdl(NewHandle(sizeof(widpt)));
    Width^^.h := 10;
    Width^^.v := 1;
    TxtPicPtr := @TxtPicRec;
    TxtPicHdl := @TxtPicPtr;
    TxtPicRec.tJus := LeftJust;
    TxtPicRec.tFlip := 0; {no flip}
    TxtPicRec.tRot := 0; {no rotation}
    TxtPicRec.tLine := 2; {1 1/2 spacing}
    GetFontInfo(fInfo);
    leading := fInfo.descent + fInfo.ascent + fInfo.leading;
    Indent := 2;
    x := -xscale / 2;
    y := a * x * x + (b * x) + c;
    hPos := integer(round(x * hor * 2 / xscale + hor));
    vPos := integer(round(-y * ver * 2 / yscale + ver));
    z1 := -b / (2 * a);
    z2 := (4 * a * c - (b * b)) / (4 * a);
    le := 2;
    tp := ver + (ver DIV 3);
    ri := 140;
    IF ri >= (hor + hor DIV 3) THEN
        ri := hor + hor DIV 3;
    bo := ver + ver - 2;
    setRect(rBox, le, tp - 14, ri, bo);
    ParagraphBegin.h := 4;
    ParagraphBegin.v := tp;

    {Graph Text}
    str1 := stringOf(-xscale DIV 2);
    str2 := stringOf(yscale DIV 2);
    str3 := stringOf(xscale DIV 2);
    str4 := stringOf(-yscale DIV 2);

    Paragraph[1] := StringOf('y=ax^2 + bx + c', chr(13));
    Paragraph[2] := StringOf('a=', a : 3 : 1, ', b=', b : 3
: 1, ', c=', c : 3 : 1, chr(13));
    Paragraph[3] := StringOf('x1=', x1 : 5 : 3, ', x2=', x2
: 5 : 3, chr(13));
    CASE result OF
        1 :
Paragraph[4] := StringOf('Two Real Roots, x1, x2', chr(13));
        2 :
Paragraph[4] := StringOf('Double Root', chr(13));
        3 :
Paragraph[4] := StringOf('Two Complex Roots ', chr(13));
        OTHERWISE
            ;
    END;
Paragraph[5] := StringOf('Slope 0 = (', z1 : 2 : 1, ',', z2 :
2 : 1, ')', chr(13));

    PenNormal;
    BackColor(Color[BackgroundColor]);
    ForeColor(Color[AxisColor]);

    {Drawing Boundry}
    PicComment(picDwgBeg, 0, NIL); {Begin MacDraw Document}
    PicComment(picGrpBeg, 0, NIL);
    PicComment(SetLineWidth, 2, Handle(Width));
    IF QDdevice = Display THEN
        FillRect(pRect, white);
    FrameRect(pRect);

    {Two Axis}
    PicComment(picGrpBeg, 0, NIL);
    moveto(0, ver);
    line(hor + hor, 0);
    moveto(hor, 0);
    line(0, ver + ver);
    PicComment(picGrpEnd, 0, NIL);

    ForeColor(Color[GraphColor]);

    {Plot Itsef}
    PicComment(picGrpBeg, 0, NIL);
```

```
      moveto(hPos, vPos);
      REPEAT
      x := x + step;
      y := a * x * x + (b * x) + c;
      hPos := integer(round(x * hor * 2 / xscale + hor));
      vPos := integer(round(-y * ver * 2 / yscale + ver));
      WITH pRect DO
          IF (hPos < right) AND (hPos > left) AND (vPos <
bottom) AND (vPos > top) THEN
              LineTo(hPos, vPos)
          ELSE
              moveto(hPos, vPos);
      UNTIL x >= xscale / 2;
      PicComment(picGrpEnd, 0, NIL);
      ForeColor(Color[1]);
      {Axis Text}
      moveto(4, ver + 14);
      DrawString(str1);
      moveto(hor - 40, 14);
      DrawString(str2);
      moveto(hor + hor - 50, ver + 14);
      DrawString(str3);
      moveto(hor - 40, ver + ver - 14);
      DrawString(str4);
      {Box }
      PicComment(picGrpBeg, 0, NIL);
      PicComment(picGrpBeg, 0, NIL);
      PicComment(SetLineWidth, 2, Handle(Width));
      IF QDdevice = Display THEN
          fillRect(rBox, white);
      frameRect(rBox);
      PicComment(picGrpEnd, 0, NIL); {of box}

      GetClip(TextClipRgn);
      ClipBox := rBox;
      ClipRect(ClipBox);
      {Box Text}
PicComment(TextBegin, sizeof(TTxtPicRec), Handle(TxtPicHdl));
      FOR LineNo := 1 TO LinesInParagraph DO
          BEGIN
              moveto(ParagraphBegin.h, ParagraphBegin.v);
              move(Indent, (LineNo - 1) * leading);
              DrawString(Paragraph[LineNo]);
          END;
      PicComment(TextEnd, 0, NIL);
      PicComment(PicGrpEnd, 0, NIL); {of Box & text}
      PicComment(PicGrpEnd, 0, NIL); {of select all objects}
      picComment(picDwgEnd, 0, NIL); {of drawing}
      SetClip(SaveClip);
      disposHandle(handle(width));
      DisposeRgn(TextClipRgn);
      DisposeRgn(SaveClip);
   END;

   PROCEDURE PlotMe;
      CONST
          Display = 1;
      VAR
          Displayrect : rect;
          pstate : PenState;
   BEGIN
      Displayrect := PicRect;
      IF PlotDocHandle^^.drawing <> NIL THEN
          BEGIN
              KillPicture(DrawingPic);
              PlotDocHandle^^.drawing := NIL;
          END;
      GetPenState(pstate);
      DrawingPic := OpenPicture(Displayrect);
      PrQDStuff(Displayrect, Display);
      ClosePicture;
      SetPenState(pstate);
      InvalRect(Displayrect); {draw picture}
      PlotDocHandle^^.drawing := DrawingPic; {save it}
   END;

   FUNCTION solveit; { : integer;}
   BEGIN
      doDialog;
      IF a <> 0 THEN
          quad(a, b, c, x1, x2, result)
      ELSE
          result := -1;
      solveit := result;
   END;

   PROCEDURE doPlot;
   BEGIN
      result := solveit;
      showWindow(PlotWindow);
      IF PlotWindow <> FrontWindow THEN
          SelectWindow(PlotWindow);
      IF result <> -1 THEN
          BEGIN
              PlotMe;
              EnableItem(myMenus[FileM], fPrint);
          END;
   END;
END.

UNIT MyPrintStuff;

INTERFACE

   USES
   ROM85, PrintTraps, PlotGlobals, Misc, myFileStuff, Solve;
   PROCEDURE doPageSet;
   PROCEDURE doPrint;
IMPLEMENTATION

   PROCEDURE PrintMe;
      CONST
          LaserWriter = 2;
      VAR
          theWorld : rect;
          pstate : penstate;
   BEGIN
      theWorld := PicRect;
      IF Option = 1 THEN
          theWorld := PicRect
      ELSE IF Option = 2 THEN
          theWorld := PageRect
      ELSE
      doMessage('Printing Rectangle Problem', '', '', '');
      GetPenState(pstate);
      PrintingPic := OpenPicture(theWorld);
      PrQDStuff(theWorld, LaserWriter);
      ClosePicture;
      SetPenState(pstate);
      DrawPicture(PrintingPic, theWorld);
      KillPicture(PrintingPic);
   END;

   PROCEDURE doPrint;
```

```
    VAR
        DoIt : boolean;
        myPrint : THPrint;
        myPrStatus : TPrStatus;
        myPrPort : TPPrPort;
        PrRect : rect;
        str1 : str255;
        temp : GrafPtr;
        numCopies : integer;
        count : integer;
        prStatus : TPrStatus;
  BEGIN (1)
    IF DrawingPic <> NIL THEN
        BEGIN (2)
          myPrint := PlotDocHandle^^.print;
          getport(temp);
          PrOpen;
          IF PrError = noErr THEN
             BEGIN (3)
                DoIt := PrValidate(myPrint);
                DoIt := PrJobDialog(myPrint);
                IF PrError <> noErr THEN
    doMessage('Printer error in job dialog', '', '', '');
                IF DoIt THEN
                   BEGIN (4)
             myPrPort := PrOpenDoc(myPrint, NIL, NIL);
             IF PrError = noErr THEN
             BEGIN (5)
                numCopies := myPrint^^.prJob.iCopies;
                FOR count := 1 TO numCopies DO
                BEGIN (6)
                   PrOpenPage(myPrPort, NIL);
                   IF PrError = noErr THEN
                   BEGIN (7)
                   { print something dummy!}
                   PageRect := myPrint^^.prInfo.rPage;
                   PrintMe;
                   END ( 7)
                   ELSE
doMessage('OpenPage error', 'cannot print this page', '', '');
                   PrClosePage(myPrPort);
                   IF PrError <> noErr THEN
doMessage('ClosePage error', 'cannot close this page', '',
'');
                END; (6)
             END ( 5)
             ELSE
          doMessage('OpenDoc error', 'cannot print', '', '');
             PrCloseDoc(myPrPort);
             IF PrError <> noErr THEN
             doMessage('CloseDoc error', '', '', '');
             IF (myPrint^^.prJob.bJDocLoop = bSpoolLoop)
AND (PrError = noerr) THEN
             PrPicFile(myPrint, NIL, NIL, NIL, prStatus);
          END; (4)
        END; (3)
        PrClose;
        setport(temp);
    END;(2)
  END; (of proc 1)


  PROCEDURE doPageSet;
    VAR
        DoIt : boolean;
        myPrint : THPrint;
  BEGIN
      myPrint := PlotDocHandle^^.print;
      PrOpen;
      IF PrError = noErr THEN
          BEGIN
             DoIt := PrValidate(myPrint);
             DoIt := PrStlDialog(myPrint);
             IF PrError <> noErr THEN
      doMessage('Printer error in style dialog', '', '', '')
             ELSE
                PageRect := myPrint^^.prInfo.rpage;
          END
      ELSE
          doMessage('Cannot perform PrOpen!', '', '', '');
      PrClose;
  END;
END. (of unit)

UNIT PlotGlobals;
INTERFACE
  USES
     ROM85, PrintTraps;
{ Global Constants }
  CONST

    (multifinder stuff)
      SysEnvTrap = $90;
      WNETrapNum = $60;    (trap number of WaitNextEvent)
      UnImplTrapNum = $9F; (trap number"unimplemented trap")
    (window constants)
      ZoomBox = 8;   (window type)
      MinWidth = 80;
      MinHeight = 80;
      mBarHeightGlobal = $BAA;
      GrayRgnLowMemGlobal = $9EE;
      sBarWidth = 16;
      rsrc = 'PLTR'; (creator bytes restype)
    (dialog stuff)
      AboutDialog = 256;
      ParamDialog = 257;
      MessageDialog = 258;
      AlertDialog = 260;
    { menu res id's}
      AppleMenu = 256;
      FileMenu = 257;
      EditMenu = 258;
      ColorMenu = 259;
      OptionMenu = 260;
    (submenus id's)
      GraphMenu = 44;
      AxisMenu = 45;
      BackgroundMenu = 46;
      MenuCount = 5;
      SubMenuStart = 6;
      TotalMenuCount = 8;
      AppleM = 1;
      FileM = 2;
      EditM = 3;
      ColorM = 4;
      OptionM = 5;
      GraphM = 6;
      AxisM = 7;
      BackGroundM = 8;
    (menu items)
      aAbout = 1;
      fPlot = 1;
      fSave = 3;
      fSaveAs = 4;
```

```
      fPageSet = 5;
      fPrint = 6;
      fQuit = 8;
      eUndo = 1;
      eCut = 3;
      eCopy = 4;
      ePaste = 5;
      eClear = 6;
      oWindowRect = 1;
      oPageRect = 2;
   {Dialog Items}
      dOK = 1;
      dA = 2;
      dB = 3;
      dC = 4;
      dSTEP = 5;
      dXSCALE = 6;
      dYSCALE = 7;
   TYPE
      Document = RECORD
            aParam : real;
            bParam : real;
            cParam : real;
            stepParam : real;
            xParam : integer;
            yParam : integer;
            drawing : PicHandle;
            print : THPrint;
            FileName : str255;
            volRefNum : integer;
         END;
      DocPtr = ^Document;
      DocHandle = ^DocPtr;
      LongAndByte = RECORD
            CASE integer OF
               1 : (
                     longView : LongInt
               );
               2 : (
                     byteView : RECORD
                           byte0 : SignedByte;
                           byte1 : Signedbyte;
                           byte2 : Signedbyte;
                           byte3 : Signedbyte;
                        END;
               );
         END;
{ Global Variables }
   VAR
   {my misc stuff}
      Finished : boolean;
      mBarHeight : Integer;
   {Multifinder stuff}
      WNE : boolean; {Multifinder friendly}
      SysEnv : boolean; {Multifinder friendly}
      theWorld : SysEnvRec; {not in LSP 1.11 }
      typeOfMac : integer;
      mouseRgn : RgnHandle; {cursor region to pass to WNE}
   {menu stuff}
      myMenus : ARRAY[1..TotalMenuCount] OF MenuHandle;
      GraphColor : integer;
      AxisColor : integer;
      BackgroundColor : integer;
      color : ARRAY[1..8] OF LongInt;
      Option : integer; {1 = windowrect, 2=pagerect}
   {rectangles}
      DragArea : Rect;  {window drag area}
      GrowArea : Rect;  {window grow area}
      Screen : Rect;    {physical screen area}
      PlotWindowRect : Rect;   {beginning window size}
      ZoomRect : Rect;  {zoomed window size}
      HCRect, VCRect, GrowRect : Rect; {scroller rects}
      PicRect : Rect; {content region of less scrollers}
      PageRect : rect;
   {dialogs stuff}
      ItemHit : integer;
      dialogflg : boolean;
   {plot stuff}
      a, b, c, x1, x2, check, step : real;
      result, xscale, yscale : integer;
      PlotWindow : WindowPtr;
      PlotWindowPeek : WindowPeek;
      PlotDocHandle : DocHandle;
      DrawingPic : PicHandle;
      PrintingPic : PicHandle;

IMPLEMENTATION

END.

UNIT Misc;

INTERFACE

   USES
      ROM85, PrintTraps, PlotGlobals;

   PROCEDURE doMessage (message0 : str255;
                                   message1 : str255;
                                   message2 : str255;
                                   message3 : str255);

IMPLEMENTATION

   PROCEDURE doMessage; {(message0 : str255)
                                   (message1 : str255)
                                   (message2 : str255)
                                   (message3 : str255))
      VAR
         dialogP : DialogPtr;
         item : integer;
   BEGIN
      ParamText(message0, message1, message2, message3);
      dialogP := GetNewDialog(MessageDialog, NIL, pointer(-
1));
      IF dialogP = NIL THEN
         BEGIN
            SysBeep(5);
            ExitToShell;
         END;
      initCursor; {change to arrow}
      ModalDialog(NIL, item);
      DisposDialog(dialogP);
   END;

END.

*Plotter.R
*

Plotter.RSRC
????????

Type PLTR = STR
   ,0
0 by Dave Kelly & Dave Smith \0Dver 4 JAN 1988
```

```
Type FREF
,128
APPL 0
,129
PICT 1

Type BNDL
,128
PLTR 0
ICN#
0 128 1 129
FREF
0 128 1 129

* —— Multifinder events ——

* bit 15 = switcher save screen
* bit 14 = accept suspend resume events
* bit 13 = switcher enable option switch
* bit 12 = can do background on null events
* bit 11 = multifinder aware
*         (activates & deactivates topmost
*          window at resume, suspend events)

Type SIZE = GNRL
   ,-1
.H
4800       ;; $4800 = bits 14,11 set
.L
128000 ;; (for 150K recomended)
.L
80000  ;; (for 80K minimum)
.I
* ——— menus ———

Type MENU
* the desk acc menu
   ,256
\14    ;;apple menu
   About Plotter…
   (-

* the file menu
   ,257
File
   Plot /P
   (-
   (Save /S
   Save as…
   Page Setup… /U
   (Print… /O
   (-
   Quit /Q

* the edit menu
   ,258
Edit
   (Undo /Z
   (-
   Cut /X
   Copy /C
   Paste /V
   Clear

* the color menu
   ,259
Color
   Graph /\1B!\2C
   Axis /\1B!\2D
   Background /\1B!\2E

* the Option menu
   ,260
Print Options
   Window Size /[
   Page Size /]

* submenus
   ,44
Graph
   Black
   White
   Red
   Green
   Blue
   Cyan
   Magenta
   Yellow

   ,45
Axis
   Black
   White
   Red
   Green
   Blue
   Cyan
   Magenta
   Yellow

   ,46
Background
   Black
   White
   Red
   Green
   Blue
   Cyan
   Magenta
   Yellow


* —— Dialogs ——
* About Box dialog...
type DLOG
   ,256
About Plotter…
100 100 250 400
visible NoGoAway
1
0
256

type DITL
   ,256
3
BtnItem Enabled
112 235 141 284
OK

StatText Disabled
10 88 141 289
Plot Demo\0D\0D++
Graphs Quadratic Equations\0D^0\0D^1\0D^2\0D^3

PicItem Disabled
10 10 96 81
128

* Plot box dialog...
type DLOG
   ,257
Plot Parameters
100 105 250 405
Visible NoGoAway
4
0
257

type DITL
   ,257
13
* ok button (default)
BtnItem Enabled
110 230 136 275
OK

* a parameter
EditText Enabled
30 15 46 60
1

* b parameter
EditText Enabled
30 100 46 145
-1

*c parameter
EditText Enabled
30 180 46 225
-6

* step parameter
EditText Enabled
80 10 96 65
.05

* xscale parameter
EditText Enabled
80 100 96 150
10

* yscale parameter
EditText Enabled
80 185 96 235
20

StatText Disabled
10 35 26 55
a

StatText Disabled
10 120 26 140
b

StatText Disabled
10 200 26 220
c
```

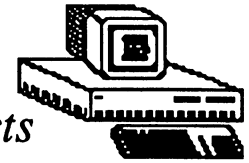

© The Definitive MacTutor, Vol. 4

```
StatText Disabled
60 10 76 65
step size

StatText Disabled
60 95 76 150
x scale

StatText Disabled
60 180 76 250
y scale

* Program Messages Dialog box...
type DLOG
   ,258
Program Messages
100 100 200 400
Visible NoGoAway
1
0
258

type DITL
   ,258
3
BtnItem Enabled
65 230 95 285
OK

StatText Disabled
15 60 85 222
^0\0D^1\0D^2\0D^3

IconItem Disabled
10 10 42 42
1

* —— Alerts ———

* Program error alerts...
type ALRT
   ,260
100 100 200 400
260
5555

type DITL
   ,260
2

BtnItem
65 230 95 285
OK

StatText Disabled
15 60 60 275
Program Problem Alert:\0D^0^1^2^3

* misc resources

Type ICN# = GNRL
  ,128 (0)
.H
0001 0000 0002 8000 0004 4000 0008 2000
0010 1000 0020 0800 0050 0400 0088 0200
0100 0100 0284 0080 0440 0240 0822 0420
1410 0810 220A 1008 4084 3F04 802A 4082
4001 8041 2003 3022 1005 C814 080E 7F8F
0412 3005 0221 0007 0140 8005 0080 6007
0040 1FE5 0020 021F 0010 0407 0008 0800
0004 1000 0002 2000 0001 4000 0000 8000
*
0001 0000 0003 8000 0007 C000 000F E000
001F F000 003F F800 007F FC00 00FF FE00
01FF FF00 03FF FF80 07FF FFC0 0FFF FFE0
1FFF FFF0 3FFF FFF8 7FFF FFFC FFFF FFFE
7FFF FFFF 3FFF FFFE 1FFF FFFC 0FFF FFFF
07FF FFFF 03FF FFFF 01FF FFFF 00FF FFFF
007F FFFF 003F FE1F 001F FC07 000F F800
0007 F000 0003 E000 0001 C000 0000 8000

Type ICN# = GNRL
  ,129 (0)
.H
0FFF FE00 0800 0300 0800 0280 0800 0240
0800 0220 0800 0210 0800 03F8 0801 0008
0880 0008 0801 0208 0840 0008 0801 0408
0820 0008 0801 0808 0810 0008 0801 1008
0AAB AAA8 0809 2008 0804 4008 0803 8008
0800 0008 0801 0008 0800 0008 0801 0008
0800 0008 0801 0008 0800 0008 0801 0008
0800 0008 0800 0008 0800 0008 0FFF FFF8
*
0FFF FE00 0FFF FF00 0FFF FF80 0FFF FFC0
0FFF FFE0 0FFF FFF0 0FFF FFF8 0FFE FFF8
0F7F FFF8 0FFE FDF8 0FBF FFF8 0FFF FBF8
0FDD 7FF8 0FFA B7F8 0FE7 DFF8 0FEF FFF8
0D74 5D58 0FB7 DBF8 0F7B BDF8 0EFD 7EF8
0FFF FFF8 0FFF FFF8 0FFF FFF8 0FFF FFF8
0FFF FFF8 0FFF FFF8 0FFF FFF8 0FFF FFF8
0FFF FFF8 0FFF FFF8 0FFF FFF8 0FFF FFF8


TYPE PICT = GNRL
   ,128
.I
891
195 254 281 325
.H
1101 A000 82A0 008E 0100 0A00 0000 0002
D002 4098 000A 00C3 00F8 00FF 0148 00C3
00FE 00FF 0145 00C3 00FE 00FF 0145 0000
02F7 0002 F700 02F7 0002 F700 02F7 0002
F700 02F7 0002 F700 02F7 0002 F700 02F7
0006 FD00 000E FC00 07FD 0001 1F80 FD00
07FD 0001 7FC0 FD00 07FD 0001 FFF0 FD00
08FE 0002 03FF FCFD 0008 FE00 0207 FFFE
FD00 09FE 0003 1FFF FF80 FE00 09FE 0003
3FFF FFE0 FE00 09FE 0003 7FFF FFF8 FE00
0A02 0000 01FE FF00 FCFE 0008 0200 0003
FDFF FE00 0A02 0000 0FFD FF00 C0FF 000B
0700 001F FFFF 3FFF E0FF 000B 0700 007F
FFFE 1FFF F8FF 000B 0700 00FF FFFE 1FFF
FCFF 000B 0100 01FE FF02 27FF FCFF 000B
0100 01FE FF02 F9FF F8FF 000B 0100 00FE
FF02 FE7F F0FF 000B 0200 003F FEFF 019F
E0FF 000B 0200 001F FEFF 01E7 C0FF 000B
0200 003F FEFF 01F9 80FF 000B 0200 0033
FEFF 01FE 80FF 000A 0200 0060 FDFF 00C0
FF00 0B07 0000 607F FFFF FCC0 FF00 0B07
0000 601F FFFF F870 FF00 0B07 0000 6007
FFFF F0F8 FF00 0B07 0000 6001 FFFF F0F8
FF00 0B07 0000 6000 FFFF F0F8 FF00 0B07
0000 6038 3FFF B050 FF00 0A06 0000 607C
0FFF 30FE 000B 0700 0060 F603 FE30 A8FF
000B 0700 0060 E301 FC30 50FF 000B 0700
0060 C000 7830 20FF 000B 0700 0060 0000
1030 88FF 000B 0200 0060 FE00 0130 50FF
000A 0200 0060 FE00 0030 FE00 0B02 0000
60FE 0001 30A8 FF00 0B07 0000 6807 0700
B050 FF00 0A06 0000 681F 8FC0 B0FE 000B
0700 006C 7FDF F1B0 A8FF 000A 0200 0067
FEFF 0030 FE00 0B09 0000 63FF FFFE 31F4
1000 0B09 0000 307F DFF0 6046 3000 0B09
0000 381F 8FC0 E045 5000 0B09 0000 1C00
0001 C044 9000 0B09 0000 0E00 0003 8044
1000 0802 0000 07FE FFFD 0009 0500 0001
FFFF FCFD 0008 FE00 0280 0004 FD00 9800
0A00 FF00 F801 1901 4800 FF00 FE01 1901
4500 FF00 FE01 1901 4500 0008 FE00 0280
0004 FD00 08FE 0002 FFFF FCFD 0008 0200
0001 FEAA FD00 0802 0000 03FE 55FD 000A
0600 0006 FEAF EA80 FE00 0A06 0000 0D83
5835 40FE 000A 0600 001B 01B0 1AA0 FE00
0A06 0000 3501 5015 50FE 000A 0600 006A
82A8 2AA8 FE00 0A06 0000 D57D 57D7 F4FE
000A 0600 01AF AAFA AC1A FE00 0A06 0003
5055 0558 0DFE 000B 0700 06A0 2A02 A80A
80FF 000B 0700 0D60 3603 5415 40FF 000B
0700 0AB0 6B06 ABEA C0FF 000B 0700 0D5F
D5FD 5555 40FF 0009 0100 0AFC AA00 C0FF
0009 0100 0DFC 5500 40FF 0009 0100 0FFC
FF00 C0FF 0002 F700 02F7 0002 F700 02F7
0002 F700 02F7 0002 F700 A000 8FA0 0083
FF

* Menu color Definitions
*
* TYPE mctb followed by ID#,
* followed by number of entries.
* (ID 0 is menu bar entry.)
* Other ID# are menu ID#.
*
* For each entry:
* 1. Menu ID number
* 2. Menu item number
* 3. RGB color 1 (3 INTEGERS)
* 4. RGB color 2 (3 INTEGERS)
* 5. RGB color 3 (3 INTEGERS)
* 6. RGB color 4 (3 INTEGERS)
* 7. filler integer


* menu bar
Type mctb = GNRL
   ,0
* number of entries
.I
1
.I
* Menu ID number & the Menu item number
* 0 & 0 for menu bar entry.
* Default title & title background =
* black on white
* Default item & item background =
* magenta on white
0 0
```

```
.H
0000 0000 0000
FFFF FFFF FFFF
FFFF 0000 FFFF
FFFF FFFF FFFF
0000

* apple menu
   ,256
.I
3
.I
* title & title background =
* Cyan on white
* default item & background=
* red on white
256 0
.H
0000 FFFF FFFF
FFFF FFFF FFFF
FFFF 0000 0000
FFFF FFFF FFFF
0000
.I
* ITEM ONE
* Mark, command, name
* and background =
* blue on white
256 1
.H
0000 0000 FFFF
0000 0000 FFFF
0000 0000 FFFF
FFFF FFFF FFFF
0000
.I
* ITEM TWO
* Mark, command, name
* and background =
* blue on white
256 2
.H
0000 0000 FFFF
0000 0000 FFFF
0000 0000 FFFF
FFFF FFFF FFFF
0000

* Graph menu
   ,44
.I
8
.I
* black on white
44 1
.H
0000 0000 0000
0000 0000 0000
0000 0000 0000
FFFF FFFF FFFF
0000
.I
* black on white
44 2
.H
0000 0000 0000
0000 0000 0000
FFFF FFFF FFFF
0000
.I
* red on white
44 3
.H
FFFF 0000 0000
FFFF 0000 0000
FFFF 0000 0000
FFFF FFFF FFFF
0000
.I
* green on white
44 4
.H
0000 FFFF 0000
0000 FFFF 0000
0000 FFFF 0000
FFFF FFFF FFFF
0000
.I
* blue on white
44 5
.H
0000 0000 FFFF
0000 0000 FFFF
0000 0000 FFFF
FFFF FFFF FFFF
0000
.I
* cyan on white
44 6
.H
0000 FFFF FFFF
0000 FFFF FFFF
0000 FFFF FFFF
FFFF FFFF FFFF
0000
.I
* magenta on white
44 7
.H
FFFF 0000 FFFF
FFFF 0000 FFFF
FFFF 0000 FFFF
FFFF FFFF FFFF
0000
.I
* yellow on white
44 8
.H
FFFF FFFF 0000
FFFF FFFF 0000
FFFF FFFF 0000
FFFF FFFF FFFF
0000

* Axis menu
   ,45
.I
8
.I
45 1
.H
0000 0000 0000
0000 0000 0000
FFFF FFFF FFFF
0000
.I
45 2
.H
0000 0000 0000
0000 0000 0000
0000 0000 0000
FFFF FFFF FFFF
0000
.I
45 3
.H
FFFF 0000 0000
FFFF 0000 0000
FFFF 0000 0000
FFFF FFFF FFFF
0000
.I
45 4
.H
0000 FFFF 0000
0000 FFFF 0000
0000 FFFF 0000
FFFF FFFF FFFF
0000
.I
45 5
.H
0000 0000 FFFF
0000 0000 FFFF
0000 0000 FFFF
FFFF FFFF FFFF
0000
.I
45 6
.H
0000 FFFF FFFF
0000 FFFF FFFF
0000 FFFF FFFF
FFFF FFFF FFFF
0000
.I
45 7
.H
FFFF 0000 FFFF
FFFF 0000 FFFF
FFFF 0000 FFFF
FFFF FFFF FFFF
0000
.I
45  8
.H
FFFF FFFF 0000
FFFF FFFF 0000
FFFF FFFF 0000
FFFF FFFF FFFF
0000

* Background menu
   ,46
.I
8
.I
46 1
.H
0000 0000 0000
0000 0000 0000
0000 0000 0000
FFFF FFFF FFFF
0000
.I
46 2
.H
0000 0000 0000
0000 0000 0000
0000 0000 0000
FFFF FFFF FFFF
0000
.I
46 3
.H
FFFF 0000 0000
FFFF 0000 0000
FFFF 0000 0000
FFFF FFFF FFFF
0000
.I
46 4
.H
0000 FFFF 0000
0000 FFFF 0000
0000 FFFF 0000
FFFF FFFF FFFF
0000
.I
46 5
.H
0000 0000 FFFF
0000 0000 FFFF
0000 0000 FFFF
FFFF FFFF FFFF
0000
.I
46 6
.H
0000 FFFF FFFF
0000 FFFF FFFF
0000 FFFF FFFF
FFFF FFFF FFFF
0000
.I
46 7
.H
FFFF 0000 FFFF
FFFF 0000 FFFF
FFFF 0000 FFFF
FFFF FFFF FFFF
0000
.I
46 8
.H
FFFF FFFF 0000
FFFF FFFF 0000
FFFF FFFF 0000
FFFF FFFF FFFF
0000
.I
```

© The Definitive MacTutor, Vol. 4

Macintosh II
Screen Dump FKEY for Color Picts



Steven C. Sheets
Chicago, IL
MacTutor Vol. 4 No. 2

In our first article, we saw how to create a plot in PICT format, complete with the old style Quickdraw color. But, since our program did not include an open command, how can we view the plot in color once it is made? If we try to snapshot the color screen, we get a beep! What we need is a way to snapshot and view our plot without having to turn off the color. Steve Sheets, from Apple's Chicago technical support office, shows us how to capture the screen on a Mac II with the color enabled.And best of all, the picture is not rotated! Once we capture our plot, we want to view it. Steve again comes to our rescue with a companion viewer DA for color PICTs.

Color Images: Saving and Loading

Since this column has given some examples of how to draw in Color on the Mac //, now would be a good time to describe how to save and load color pictures. While a specific file format can always be defined for an application, it is convenient to be able to read generic file formats, so that various applications can share and port data. When discussing graphics, the two generic file formats have always been MacPaint file format and PICT file format. MacPaint format has always been considered the generic Bitmap method of saving Mac graphics, while PICT format is the associated with object oriented graphics programs (ie. MacDraw).

With the introduction of Color Quickdraw and the Mac //, things have changed slightly. Unfortunately MacPaint file format can not be expanded to cover PixMap graphics (ie. multiple colors per pixel). Nor is there any other PixMap file format that has been widely accepted by developers (though there is a need). On the other hand, PICT file format is nothing but an extension of a Quickdraw Picture Handle. With the introduction of the Mac //, Quickdraw Pictures have been expanded to support color. This article will talk about the new Color Quickdraw Picture format, the PICT file format, and will touch briefly on using the Palette Manager to display a Picture in the Color needed. The example sources, ColorImage DA and ColorImage FKEY, will demonstrate how to save and read PICT file.

Quickdraw Picture Format

A Quickdraw Picture is a data structure that contains a sequence of Quickdraw commands. To create a Picture, a program defines a Picture Frame (ie, a rectangle and data structure) with the OpenPicture function. OpenPicture returns a handle to the Picture data structure being created. At this point, the program can call any Quickdraw command. Instead of being drawn on the current grafport, the command is stored in the Picture for later use. A ClosePicture command needs to be issued to tell Quickdraw to stop saving the commands into the Picture Handle. Once the program has a Picture, it can display it using the DrawPicture call. This will draw the Picture (ie. all the Quickdraw commands saved in the Picture) into a destination rectangle. If the destination rectangle is not the same size as the picture frame, the picture will be expanded or reduced to fit the destination rectangle. The Plotter program in this issue illustrates this traditional use of the Quickdraw picture.

The Picture handle points to a data structure which contains the Picture Size (Integer picSize), the Picture Frame (Rect picFrame) and various Opcodes describing the Quickdraw calls. Always ignore picSize. With the expansion of the Memory Manager so that it can handle sizes beyond 32K, picSize is now ignored by Quickdraw. Instead, the picture is played out until the end. If the Picture size is needed, call GetHandleSize. Generally beyond picSize and picFrame, a program should not read the data structures directly. The internal structures beyond this point are documented in *Inside Macintosh, Vol 5* and the various tech notes cited in the Pascal column.

An image stored and drawn as a Picture (using DrawPicture) is much more compressed than the same image drawn using individual Quickdraw commands in the code. Also the program can pass a Picture through the clipboard and Scrapbook to other Applications. Notice while the Picture is considered an Object Graphic type of data structure, one of the commands it stores is CopyBits. A Picture can thus have BitMapped graphics in it.

Version 1 vs Version 2 Format

With the introduction of the Mac //, the Picture Format has been extended. The old Picture format (called Version 1 Format) contains only the non-Color Quickdraw commands. The newer Picture format (called Version 2 Format) contains all the Color Quickdraw commands. On a Mac //, which Format is created with an OpenPicture call depends on the current GrafPort. If an OpenPicture call is made when the current GrafPort is a Color GrafPort, the Picture created is a Version 2 format. If a call is made on a normal GrafPort, a Version 1 Picture is created. Obviously non-Mac // computers can not create a Version 2 Picture. However, due to a patch in System 4.1 and later, an Enhanced Mac 512K, a MacPlus or a Mac SE can display a Version 2 Picture (not in color, of course).

While checking the Opcodes (Quickdraw picture commands) directly is not a good idea, a program can identify which version of Picture it has by checking the first integer past the picSize and picFrame. If this integer is 4353 ($1101), the Picture is Vers. 1. If the integer is 17 ($0011), the Picture is Vers. 2.

Just as a Version 1 Picture can have bitmap graphics in it, a Version 2 Picture can have a PixMap in it. Any CopyBits calls

made following an OpenPicture, but before a ClosePicture call, will store the data into the Picture. If the source was a PixMap instead of a BitMap, the Picture handle will have both the bit data (compressed format) and a color table (Color Look Up Table or CLUT) describing the colors used in the PixMap.

PICT File Format

Knowing the Picture format, the PICT File Format is simple. A PICT file (ie. one that has the Filetype 'PICT') consists of a data fork with a 512 byte header followed by a picture data structure. The header is usually specific to the application that made it (most programs ignore it). After the header, the picture data contains picSize, picFrame and then the variable length Opcodes. As stated above, do not use picSize to determine how much data follows. Instead, check the size of the file with a File I/O call. Subtract 512 and that should be the size of the handle that the data should be read into.

ColorImage FKEY Code

The ColorImage FKEY was designed to save the current screen to file, similar to the Screen Save FKEY (Command-Shift-3). Instead of a MacPaint file being created, a PICT file is created instead. A CopyBits call on the current screen is used when creating the picture handle, which is then saved to the file.

The FKEY will work on any Macintosh. If the Macintosh has Color Quickdraw (ie. Macintosh //), the Color Window Manager GrafPort is temporarily set before creating the picture handle. Thus a version 2 PICT file will be created and any color images on the screen will be saved. On a non-Color Quickdraw Macintosh (ie. Mac SE and before), the normal Window Manager Port is used and a version 1 PICT file is created. If the Caps Lock key is pressed, the entire screen is saved to a file. If the Caps Lock key is not pressed and a front window exists, only the current window is saved to a file. The picture is not saved to a set file name like the Screen Save FKEY, instead the user is prompted for the file name. The image is also not rotated on a Mac II monitor. Thus you won't need to edit your screen dumps before using them in your documentation, nor will you need to turn off your color interface as is the case with the old style shift-cmd-3.

Remember, if the Screen was a PixMap instead of a Bitmap, the CLUT will be saved into the PICT file along with the compressed bit data. Color PICTs are amazingly packed files.

Palette Manager

Reading a PICT file, however, is only half the job of displaying a color image correctly. If the Picture contained Color Quickdraw commands (for example, a Color Window saved with ColorImage FKEY), the user will want to see the Picture drawn again using the original colors. For this, the Palette Manager needs to be used.

The Palette Manager establishes the color environment of the Mac //. It uses the Color Manager (which controls what colors are used) to select the best colors for a given display. The idea is that each window can have a Color Palette associated with it. These are the Colors that the window wants the screen to be showing, in order to view the window at it's best. Unless there are to many windows needing to many colors, each window gets it's Palette. If there are not enough colors, highest priority goes to the top most window. Every time the top most window changes (and it has an associated Palette), the colors may change to give the new window it's best appearance. The Palette Manager will then issue an update event for all Palette windows so that they can be redrawn to their best appearance in the new colors.

While an Application could use the Color Manager directly and bypass the Palette Manager, this is not recommended. Such programming would fall apart in a color Multiple task environment. Remember, Multifinder is not the only such environment. There are also color DAs like ColorImage.

Thus the DA takes the Picture (if it is Version 2), searches for a CLUT, and tries to create a Color Palette from it. The program does all this by temporarily replacing the low-level quickdraw procs (specifically StdBits) with one that checks for a PixMap, then transfers the RGB colors into a temporary buffer. The low-level routines are replaced and the buffer is converted to a Color Palette. Then the Palette is attached to the window so every time that particular window becomes the top most, the screen colors are set to the Palette. This also causes an update of the window so the Picture can be shown in it's best colors.

ColorImage DA Code

The ColorImage Desk Accessory contains the complementing code to Color FKEY, and more. The DA creates a window (Color Window if the CPU has Color Quickdraw) to display Pictures in. The user can then select a PICT file and have it read into a Picture handle. Since a user may want to know what colors are being used for a specific Picture, the DA allows the users to switch between looking at the Picture or the current Palette.

Think of two ColorImage DA's that are loaded into the system. Each opens a Picture with a radically different CLUT (one is a 256 color RGB Picture while the other is a 256 shade Gray Scaled one). When the RGB Picture is clicked on, it becomes the front window. The Palette Manager sets the 256 RGB colors (assuming the Color Video Card has 8 Bits per Pixel). Both pictures are completely redrawn using the current screen colors. Notice the Palette Manager caused the update; the program does not have to keep track. The RGB Picture appears perfect, while the Gray Picture (which can be seen behind the front window) appears as best as it can using the current screen colors. Then the user wants to look at the Gray Picture. He clicks on that window. It becomes front window, and the Palette Manager sets the current screen to 256 shades of gray. The windows are redrawn. The Gray Picture appears perfectly, while the RGB one appears as best it can using these new gray tones.

Last Comments

The ColorImage FKEY and Desk Accessory were created with LightSpeed Pascal (LSP). Macintosh Programming Workshop (MPW) has it's points. It is an extremely powerful development system perfect for large projects. Being from Apple, MPW is the first development system that has any new Trap

Listings or new code interfaces. LSP only recently has released the Mac // libraries and interfaces. MPW is also the best Object Pascal system around. However LSP may be the best environment to develop code segments (ie. DA, FKEYs, Window or Control Panel Definition Procedures) for the Macintosh. Both code examples were tested from inside a test program (a simple program that called the FKEY and a modified version of LSP DAShell). This decreases development time since the various codes could be tested without having to insert the DA or FKEY into the System file. The code could even be stepped through one line at a time, perhaps the best feature of LSP!

This program requires the Palette manager. When LS Pascal version 1.11 was released in September, Think did not include the Palette manager, nor the MultiFinder routines. We are still waiting for both Think and Borland to upgrade their products to MPW version 2.0 interfaces. You can add the Palette manager by taking the MPW 2.0 interface and adding it to your project. Remove the USES MemTypes and replace with USES ColorQuickdraw instead. Then add "Implementation" to the end of the file and your in business.

ColorImage DA could be expanded to check not only the PixMap's CLUT in a Picture, but could check other Color Quickdraw commands. Since the DA was designed to work with the FKEY, checking the CLUT is all that is needed to function. The program could also check for colors that are in the CLUT, but are NOT present in the PixMap. Such colors are not displayed and need not be in the Palette.

Yet to come... a full blown explanation of the Palette Manager and Palette Animation!

```
{ ColorImage DA / FKEY by Steve Sheets 12/87   }
{ Saves the current Screen to a PICT file.  If Cap Lock }
{is down, saves the entire screen.  If the key is not & a}
{ Window exists, saves the window.}

UNIT UColorImageFKEY;

INTERFACE

   PROCEDURE main;

IMPLEMENTATION

{  Returns the Color Window Manager Port.  }
   PROCEDURE CGetWMgrPort (VAR CWPort : GrafPtr);
   INLINE
      $AA48;

{  Does this CPU have color Quickdraw?  }
   FUNCTION ColorQDExists : boolean;
      CONST
         ROM85Loc = $28E;
         TwoHighMask = $C000;
      TYPE
         WordPtr = ^INTEGER;
      VAR
         Wd : WordPtr;
   BEGIN
      Wd := POINTER(ROM85Loc);
      ColorQDExists := (BitAnd(Wd^, TwoHighMask) = 0);
   END;

{  Is Cap Lock key down?  }
   FUNCTION CapLockDown : BOOLEAN;
      VAR
         theMap : KeyMap;
   BEGIN
      GetKeys(theMap);
      CapLockDown := BitTst(@theMap[1], 30);
   END;

{  Given a flag saying color Quickdraw exists, returns}
{the Handle of a Picture and a flag saying if the window }
{was saved or if the entire screen was.  }
   PROCEDURE GetPic (VAR thePic : PicHandle;
                              VAR IsWindow : BOOLEAN;
                              QDExists : BOOLEAN);
      TYPE
         BitMapPtr = ^BitMap;
         BitMapHdl = ^BitMapPtr;
      VAR
         srcRect, destRect, mapRect : Rect;
         OldPort, WPort : GrafPtr;
         clip, vis : RgnHandle;
         FWin : WindowPtr;
         PixH : BitMapHdl;
         tempMap : BitMap;
   BEGIN
      GetPort(OldPort);

{ If Color Quickdraw exists, use the Color Window Manager
{ PixMap. Else use the normal Window Manager BitMap. }
      IF QDExists THEN
         BEGIN
            CGetWMgrPort(WPort);
            PixH := BitMapHdl(WPort^.portBits.baseAddr);
            mapRect := PixH^^.bounds;
         END
      ELSE
         BEGIN
            GetWMgrPort(WPort);
            mapRect := WPort^.portBits.bounds;
         END;

{  Chose to save the Window or entire Screen.  }
      FWin := FrontWindow;
      IsWindow := (NOT CapLockDown) AND (FWin <> NIL);
      IF IsWindow THEN
         BEGIN
            SetPort(FWin);
            srcRect := FWin^.portRect;
            LocalToGlobal(srcRect.topLeft);
            LocalToGlobal(srcRect.botRight);
            OffSetRect(srcRect, mapRect.left, mapRect.top);
         END
      ELSE
         srcRect := mapRect;
      destRect.left := 0;
      destRect.right := srcRect.right - srcRect.left;
      destRect.top := 0;
      destRect.bottom := srcRect.bottom - srcRect.top;

{  Resize the regions so they do not get in the way. }
      SetPort(WPort);
      vis := WPort^.visRgn;
      clip := WPort^.clipRgn;
      WPort^.visRgn := NewRgn;
      WPort^.clipRgn := NewRgn;
      SetRectRgn(WPort^.visRgn, -30000, -30000, 30000, 30000);
   SetRectRgn(WPort^.clipRgn, -30000, -30000, 30000, 30000);

{  Create Picture, copy map to it and close the Picture. }
      thePic := OpenPicture(destRect);
      CopyBits(WPort^.portBits, WPort^.portBits, srcRect,
destRect, srcCopy, NIL);
      ClosePicture;

{  Reset the Regions and Port.}
```

```
      DisposeRgn(WPort^.visRgn);
      DisposeRgn(WPort^.clipRgn);
      WPort^.visRgn := vis;
      WPort^.clipRgn := clip;

      SetPort(OldPort);
   END;

{  Pick a PICT file to save the Picture into (with correct
title line).  }
   PROCEDURE PickFile (VAR theReply : SFReply;
                           IsWindow : BOOLEAN;
                           QDExists : BOOLEAN);
      VAR
         Pt : Point;
         S1, S2 : str255;
   BEGIN
      Pt.v := 60;
      Pt.h := 60;
      IF QDExists THEN
         S1 := 'Save Color '
      ELSE
         S1 := 'Save ';
      IF IsWindow THEN
         S2 := 'Window PICT:'
      ELSE
         S2 := 'Screen PICT:';
      SFPutFile(Pt, CONCAT(S1, S2), '', NIL, theReply);
   END;

{  Given a Picture handle and a reference to a file, create a
PICT file and save the Picture into it.  }
   PROCEDURE SaveColorImage (thePic : PicHandle;
                             theReply : SFReply);
      TYPE
         BufType = PACKED ARRAY[1..128] OF LongInt;
      VAR
         theErr : OSErr;
         OK : BOOLEAN;
         FileRef : INTEGER;
         BufSize : LongInt;
         Buf : BufType;
         count : integer;
   BEGIN
      OK := FALSE;
      FileRef := 0;
      theErr := FSDelete(theReply.fname, theReply.vRefNum);
      IF (theErr = noErr) OR (theErr = fnfErr) THEN
      BEGIN
         theErr := Create(theReply.fname, theReply.vRefNum,
'MDRW', 'PICT');
         IF (theErr = noErr) THEN
         BEGIN
         theErr := FSOpen(theReply.fname, theReply.vRefNum,
FileRef);
         IF (theErr = noErr) THEN
            BEGIN
               FOR count := 1 TO 128 DO
                  Buf[count] := 0;
               BufSize := 512;
               theErr := FSWrite(FileRef, BufSize, @Buf);
               IF (theErr = noErr) THEN
               BEGIN
                  BufSize := GetHandleSize(Handle(thePic));
                  HLock(Handle(thePic));
         theErr := FSWrite(FileRef, BufSize, Ptr(thePic^));
                  HUnLock(Handle(thePic));
                  IF (theErr = noErr) THEN
                  BEGIN
                     theErr := FSClose(FileRef);
                     FileRef := 0;
                     IF (theErr = noErr) THEN
                     BEGIN
                  theErr := FlushVol(NIL, theReply.vRefNum);
                        OK := TRUE;
                     END;
                  END;
               END;
            END;
         END;
         IF NOT OK THEN
            SysBeep(60);
         IF (FileRef <> 0) THEN
            theErr := FSClose(FileRef);
   END;

{Main FKEY routine:  Check if this CPU has Color Quickdraw}
{(used by all routines).  Create the Picture Handle and Pick}
{ a File.  If successful, save the Picture into the file and}
{ kill the picture.}
   PROCEDURE main;
      VAR
         thePic : PicHandle;
         theReply : SFReply;
         IsWindow : BOOLEAN;
         QDExist : BOOLEAN;
   BEGIN
      QDExist := ColorQDExists;
      GetPic(thePic, IsWindow, QDExist);
      PickFile(theReply, IsWindow, QDExist);
      IF theReply.good THEN
         SaveColorImage(thePic, theReply);
      KillPicture(thePic);
   END;
END.

{  ColorImage DA by Steve Sheets 12/87  }

{  Displays a PICT file.  Checks if the PICT has a Color}
{ PixMap. If so, use the Palette Manager to display the PICT}
{ in the correct colors.  }

UNIT xColorImageDA;

INTERFACE

   USES
      ColorQuickDraw, PaletteMgr, ColorWindowMgr;

   FUNCTION Main (theDCE : DCtlPtr;
                           IOPB : ParmBlkPtr;
                           driveCall : Integer) : OSErr;

IMPLEMENTATION

   CONST
      DriverOpen = 0;
      DriverPrime = 1;
      DriverControl = 2;
      DriverStatus = 3;
      DriverClose = 4;

      MaxPal = 512;

      dCtlEnable = 10;

      kNum = 16;
      kBox = 20;
      kHst = 30;
      kVst = 40;
      kVoff = 40;

      kAbout = 1;
      kLoadPic = 3;
      kShowPic = 5;
      kShowPal = 6;
```


© The Definitive MacTutor, Vol. 4

```
TYPE
   BufType = RECORD
         Counter : integer;
         CLUTnum : longint;
         B : ARRAY[1..MaxPal] OF RGBColor;
      END;
   BufPtr = ^BufType;
   BufHdl = ^BufPtr;
   imageData = RECORD
         imagePic : PicHandle;
         imagePal : PaletteHandle;
         imageID : integer;
         imageWindow : WindowPtr;
         imageMenu : MenuHandle;
         picRect, palRect : Rect;
         ColorFlag, ShowPicFlag : boolean;
         numPal : integer;
         noPicstr, nameStr : str255;
      END;
   imagePtr = ^imageData;
   imageHandle = ^imagePtr;
   MyCSpecArray = ARRAY[0..MaxPal] OF ColorSpec;
   MyCTabHandle = ^MyCTabPtr;
   MyCTabPtr = ^MyColorTable;
   MyColorTable = RECORD
         ctSeed : LONGINT;
         transIndex : INTEGER;
         ctSize : INTEGER;
         ctTable : MyCSpecArray;
      END;

{  Resize the window W to the size R.  Updates the entire}
{ window   (window must be the current Grafport).  }
   PROCEDURE ReSize (W : WindowPtr;R : rect);
   BEGIN
      SizeWindow(W, R.right, R.bottom, false);
      InvalRect(R)
   END;

{  Low level Quickdraw Proc.  Instead of drawing a} {Bitmap/
Pixmap, grabs the Color Lookup Table, and passes}
{the RGB colors into the    buffer (stored in the Window)
{Ref Con). }
   PROCEDURE BitInfo (VAR srcBits : PixMap;
                      VAR srcRect, dstRect : Rect;
                      mode : integer;
                      maskRgn : RgnHandle);
      VAR
         tempW : WindowPtr;
         theBuf : BufHdl;
         tempC : MyCTabHandle;
         n, count : integer;
   BEGIN
   GetPort(tempW);
   theBuf := POINTER(GetWRefCon(tempW));
   {  Make sure it is a PixMap.  }
   IF (srcBits.rowBytes < 0) THEN
      BEGIN
      tempC := POINTER(srcBits.pmTable);
      {  Make sure this is a new CLUT.  }
      IF (tempC^^.ctSeed <> theBuf^^.CLUTnum) THEN
         BEGIN
         {  Cycle through all colors, stuffing into Buf.}
            theBuf^^.CLUTnum := tempC^^.ctSeed;
            n := theBuf^^.Counter;
            IF tempC^^.ctSize >= 0 THEN
               FOR count := 0 TO tempC^^.ctSize DO
               BEGIN
                  IF n < MaxPal THEN
                     BEGIN
                        n := n + 1;
               theBuf^^.B[n] := tempC^^.ctTable[count].rgb;
                     END;
               END;
            theBuf^^.Counter := n;
         END;
      END;
   END;

{  Given a window and a Picture Handle, return a}
{ Palette Handle and Palette number of the colors used}
{ in the Picture Handle.   }
   FUNCTION GetPal (theW : WindowPtr;
                    thePic : PicHandle;
                    VAR thePal : PaletteHandle;
                    VAR theNum : integer) : boolean;
      TYPE
         myPicHdl = ^myPicPtr;
         myPicPtr = ^myPicData;
         myPicData = RECORD
               P   : Picture;
               ID  : integer;
            END;
      VAR
         count : integer;
         theBuf : BufHdl;
         tempPtr : myPicHdl;
         tempProc : CQDProcs;
         oldProc : QDProcsPtr;
         oldRef : longint;
         dummy : boolean;
         tempP : GrafPtr;
   BEGIN
      dummy := false;
      {  If this is a Handle,}
      IF thePic <> NIL THEN
         BEGIN
         {  And if it is a color Picture. }
            tempPtr := POINTER(thePic);
            IF tempPtr^^.ID = $0011 THEN
               BEGIN
               {  Create the Buffer.  }
               theBuf := POINTER(NewHandle(SizeOf(BufType)));
               IF (theBuf <> NIL) THEN
                  BEGIN
                  { Clear Buffers counters.  }
                     theBuf^^.Counter := 0;
                     theBuf^^.CLUTnum := -1;
{  Save the old Window RefCon, and stuff Buffer into RefCon.}
                     oldRef := GetWRefCon(theW);
                     SetWRefCon(theW, ord4(theBuf));
{  Save the old low level QD Procs, create new ones (including
BitInfo). }
                     oldProc := POINTER(theW^.grafProcs);
                     SetStdCProcs(tempProc);
                     tempProc.bitsProc := @BitInfo;
                     theW^.grafProcs := @tempProc;
{  Draw the Picture (grabing the CLUT and RGB Colors).  }
                     GetPort(tempP);
                     SetPort(theW);
                     DrawPicture(thePic, thePic^^.picFrame);
                     SetPort(tempP);
                     theW^.grafProcs := POINTER(oldProc);
                     SetWRefCon(theW, oldRef);
{  If there were any colors, create a Color Palette.}
                     IF theBuf^^.Counter > 0 THEN
                     BEGIN
   thePal := NewPalette(theBuf^^.Counter, NIL, pmTolerant, 0);
                        IF thePal <> NIL THEN
                           BEGIN
                              dummy := true;
                              theNum := theBuf^^.Counter;
                              FOR count := 1 TO theNum DO
         SetEntryColor(thePal, count - 1, theBuf^^.B[count]);
                           END;
                     END
                     ELSE
                        thePal := NIL;
```

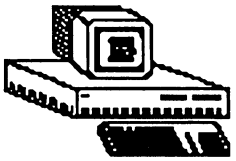
```
                    DisposHandle(POINTER(theBuf));
                END;
            END;
        END;
    GetPal := dummy;
END;

{  Given references to a PICT file, read the PICT file into a
PICT handle.  }
FUNCTION GetPic (theReply : SFReply;
                 VAR thePic : PicHandle) : BOOLEAN;
    VAR
        OK : BOOLEAN;
        FileRef : INTEGER;
        BufSize : LongInt;
        theE : OSerr;
        tempPic : PicHandle;
BEGIN
    OK := FALSE;
    FileRef := 0;
    tempPic := NIL;
    theE := FSOpen(theReply.fname, theReply.vRefNum,
FileRef);
    IF (theE = noErr) THEN
    BEGIN
        theE := GetEOF(FileRef, BufSize);
        IF (theE = noErr) THEN
        BEGIN
            BufSize := BufSize - 512;
            theE := SetFPos(FileRef, fsFromStart, 512);
            IF (theE = noErr) THEN
            BEGIN
                tempPic := PicHandle(NewHandle(BufSize));
                theE := MemError;
                IF (theE = noErr) THEN
                BEGIN
                    HLock(Handle(tempPic));
        theE := FSRead(FileRef, BufSize, POINTER(tempPic^));
                    HUnLock(Handle(tempPic));
                    OK := (theE = noErr);
                    IF OK THEN
                        thePic := tempPic;
                END;
            END;
        END;
    END;
    IF (FileRef <> 0) THEN
        FileRef := FSClose(FileRef);
    IF NOT OK THEN
        BEGIN
            SysBeep(1);
            IF tempPic <> NIL THEN
                DisposHandle(Handle(tempPic));
        END;
    GetPic := OK;
END;

{  Load a PICT into the DA.   }
PROCEDURE LoadPic (theDCE : DCtlPtr);
    VAR
        theImage : ImageHandle;
        tempReply : SFReply;
        P : point;
        theList : SFTypeList;
        S : str255;
BEGIN
    theImage := ImageHandle(theDCE^.dCtlStorage);
    WITH theImage^^ DO
        BEGIN
        {  Select a File.   }
        P.h := 60;
        P.v := 60;
        theList[0] := 'PICT';
        SFGetFile(P, '', NIL, 1, theList, NIL, tempReply);
        IF tempReply.good THEN
        BEGIN
        {If a files is selected, release the old Picture and
Palette (if any).}
            IF imagePic <> NIL THEN
            BEGIN
                DisposHandle(Handle(imagePic));
                imagePic := NIL;
                picRect.bottom := kBox * kNum;
                picRect.right := kBox * kNum;
            END;
            IF imagePal <> NIL THEN
            BEGIN
                DisposePalette(imagePal);
                numPal := 0;
                imagePal := NIL;
            END;
            {  Prepare to set Windows name to 'ColorImage'.}
            S := nameStr;
            {  Read the Picture.}
            IF GetPic(tempReply, imagePic) THEN
            BEGIN
   { If PICT read correctly, resize the window & set the
Window name to 'ColorImage : (filename)'.}
            picRect.bottom := imagePic^^.picFrame.bottom -
imagePic^^.picFrame.top;
            picRect.right := imagePic^^.picFrame.right -
imagePic^^.picFrame.left;
            S := CONCAT(S, ': ', tempReply.fname);
            {If this is Color Quickdraw CPU.}
            IF ColorFlag THEN
            BEGIN
                {  Get Palette from Picture.}
                IF GetPal(imageWindow, imagePic, imagePal,
numpal) THEN
                BEGIN
   { If successful, set the Palette to the Image Window &
resize the Pal rect.}
                SetPalette(imageWindow, imagePal, true);
                IF numPal > kNum THEN
                    palRect.right := palRect.left + (kNum *
kBox)
                ELSE
                    palRect.right := numPal * kBox;
                palRect.bottom := (((numPal - 1) DIV kNum)
+ 1) * kBox;
                END;
            END;
        END;

{  Set the Menu correctly, set so the Picture is displayed
first, resize the window, and set the Window Title. }
        IF imagePic <> NIL THEN
            EnableItem(imageMenu, kShowPic)
        ELSE
            DisableItem(imageMenu, kShowPic);
        IF imagePal <> NIL THEN
            EnableItem(imageMenu, kShowPal)
        ELSE
            DisableItem(imageMenu, kShowPal);
        ShowPicFlag := true;
        ReSize(imageWindow, picRect);
        SetWTitle(imageWindow, S);
        END;
    END;
END;

{  Does this CPU have color Quickdraw?  }
FUNCTION ColorQDExists : boolean;
    CONST
        ROM85Loc = $28E;
        TwoHighMask = $C000;
    TYPE
        WordPtr = ^INTEGER;
```

```
      VAR
         Wd : WordPtr;
   BEGIN
      Wd := POINTER(ROM85Loc);
      ColorQDExists := (BitAnd(Wd^, TwoHighMask) = 0);
   END;

{  Open the DA.  Set data structures, Menu, window,  etc.}
   FUNCTION Open (theDCE : DCtlPtr) : OSErr;
      VAR
         tempPort : Grafptr;
         theImage : ImageHandle;
         tmpRect : Rect;
         insideWindow : WindowPeek;
   BEGIN
      getport(tempPort);
      IF theDCE^.dCtlStorage = NIL THEN
         BEGIN
         theDCE^.dCtlStorage := NewHandle(SizeOf(imageData));
         HLock(theDCE^.dCtlStorage);
         theImage := ImageHandle(theDCE^.dCtlStorage);
         WITH theDCE^, theImage^^ DO
               BEGIN
                  imageID := dCtlRefNum;
                  IF imageID > 0 THEN
                     imageID := -imageID;
                  imageID := $C000 + 32 * (-1 - imageID);
                  imageMenu := GetMenu(imageID);
                  IF dCtlRefNum < 0 THEN
                     BEGIN
                        imageMenu^^.menuID := imageID;
                        theDCE^.dctlMenu := imageID;
                     END
                  ELSE
                     BEGIN
                        imageMenu^^.menuID := -imageID;
                        theDCE^.dctlMenu := -imageID;
                     END;
                  imagePic := NIL;
                  imagePal := NIL;
                  numPal := 0;
                  DisableItem(imageMenu, kShowPic);
                  DisableItem(imageMenu, kShowPal);
                  picRect.top := 0;
                  picRect.left := 0;
                  picRect.bottom := kBox * kNum;
                  picRect.right := kBox * kNum;
                  palRect := picRect;
                  ShowPicFlag := true;
                  ColorFlag := ColorQDExists;
                  GetIndString(nameStr, imageID, 1);
                  GetIndString(nopicStr, imageID, 2);
                  tmpRect := picRect;
                  OffSetRect(tmpRect, 0, kVoff);
                  IF ColorFlag THEN
                     imageWindow := NewCWindow(NIL, tmpRect,
nameStr, true, rDocProc, Pointer(-1), True, 0)
                  ELSE
                     imageWindow := NewWindow(NIL, tmpRect,
nameStr, true, rDocProc, Pointer(-1), True, 0);
                  SetPort(imageWindow);
                  insideWindow := WindowPeek(imageWindow);
                  insideWindow^.windowKind := dCtlRefNum;
                  dCtlWindow := Ptr(imageWindow);
               END;
            HUnLock(theDCE^.dCtlStorage);
         END
      ELSE
         BEGIN
            SelectWindow(WindowPtr(theDCE^.dCtlWindow));
            SetPort(WindowPtr(theDCE^.dCtlWindow));
         END;
      SetPort(tempPort);
      Open := noErr;
   END;

{  Do nothing Prime routine.  }
   FUNCTION Prime (theDCE : DCtlPtr;IOPB : ParmBlkPtr) :
OSErr;
   BEGIN
      Prime := noErr;
   END;

{  Do the Control routine (ie. Events, Menus, etc.).}
   FUNCTION Control (theDCE : DCtlPtr;IOPB : ParmBlkPtr) :
OSErr;
      CONST
         accEvent = 64;
         accMenu = 67;
         accUndo = 68;
         accCut = 70;
         accCopy = 71;
         accPaste = 72;
         accClear = 73;
      VAR
         theEvent : EventRecord;
         theImage : ImageHandle;
         aWindow : WindowPtr;
         tempRect : Rect;
         tempPort : Grafptr;
         count : integer;
   BEGIN
   getport(tempPort);
   HLock(theDCE^.dCtlStorage);
   theImage := ImageHandle(theDCE^.dCtlStorage);
   WITH theImage^^ DO
   BEGIN
      SetPort(imageWindow);
      CASE IOPB^.csCode OF
      accEvent :
         BEGIN
            BlockMove(IOPB^.ioMisc, @theEvent,
SizeOf(theEvent));
            CASE theEvent.what OF
            {Update event:  Show the Picture or the Palette.}
            updateEvt :
               BEGIN
               BeginUpdate(WindowPtr(theEvent.message));
               SetPort(WindowPtr(theEvent.message));
               IF ShowPicFlag THEN
               BEGIN
                  EraseRect(picRect);
                  IF imagePic <> NIL THEN
                     DrawPicture(imagePic, picRect)
                  ELSE
                     BEGIN
                        MoveTo(kHst, kVst);
                        DrawString(noPicStr);
                     END;
               END
               ELSE IF ColorFlag THEN
                  BEGIN
                     EraseRect(palRect);
                     IF imagePal <> NIL THEN
                        FOR count := 0 TO NumPal - 1 DO
                        BEGIN
                  PmForeColor(count);
                  tempRect.top := (count DIV kNum) * kBox;
                  tempRect.left := (count MOD kNum) * kBox;
                  tempRect.bottom := tempRect.top + kBox;
                  tempRect.right := tempRect.left + kBox;
                  PaintRect(tempRect);
                        END;
                  END;
               EndUpdate(WindowPtr(theEvent.message));
               END;
            {  Activate Event:  Add or remove the Menu.  }
            activateEvt :
```

```
        IF BitAND(theEvent.modifiers, activeFlag) <> 0 THEN
           BEGIN
           IF Ord(theEvent.message) = Ord(imageWindow) THEN
              BEGIN
                 InsertMenu(imageMenu, 0);
                 DrawMenuBar;
              END;
           END
        ELSE IF Ord(theEvent.message) = Ord(imageWindow)
THEN                                (De-activate event)
           BEGIN
              DeleteMenu(imageMenu^^.menuID);
              DrawMenuBar;
           END;
           {  Mouse Down:  Drag the window.     }
        mouseDown :
           IF FindWindow(theEvent.where, awindow) IN
[inSysWindow, inContent] THEN
           BEGIN
           SetRect(tempRect, -30000, -30000, 30000, 30000);
           DragWindow(awindow, theEvent.where, tempRect);
           END;
        keyDown, autoKey :
           SysBeep(1);
        OTHERWISE
        END;
     END;
{  Menus:  Do the About Alert, Load a Picture or switch
between the Picture or Palette.   }
        accMenu :
           CASE IOPB^.CSParam[1] OF
              kAbout :
                 count := Alert(imageID, NIL);
              kLoadPic :
                 LoadPic(theDCE);
              kShowPic :
                 IF NOT ShowPicFlag THEN
                    BEGIN
                       ShowPicFlag := true;
                       ReSize(imageWindow, picRect);
                    END;
              kShowPal :
                 IF ShowPicFlag THEN
                    BEGIN
                       ShowPicFlag := false;
                       ReSize(imageWindow, palRect);
                    END;
              OTHERWISE
           END;
           accUndo, accCut, accCopy, accPaste, accClear :
              SysBeep(1);
           OTHERWISE
        END;
     END;
     HUnLock(Handle(theImage));
     SetPort(tempPort);
     Control := noErr;
  END;
{  Do nothing Status routine. }
  FUNCTION Status (theDCE : DCtlPtr;IOPB : ParmBlkPtr) :
OSErr;
  BEGIN
     Status := noErr;
  END;

{  Close Routine:  Clear the Menu, Window, Palette, Picture
and Data Structure.  }
  FUNCTION Close (theDCE : DCtlPtr;IOPB : ParmBlkPtr) :
OSErr;
     VAR
        theImage : ImageHandle;
  BEGIN
     HLock(theDCE^.dCtlStorage);
     theImage := ImageHandle(theDCE^.dCtlStorage);
     WITH theImage^^ DO
        BEGIN
           DeleteMenu(imageMenu^^.menuID);
           ReleaseResource(Handle(imageMenu));
           DrawMenuBar;
           IF imagePal <> NIL THEN
              DisposePalette(imagePal);
           IF imagePic <> NIL THEN
              DisposHandle(Handle(imagePic));
           DisposeWindow(imageWindow);
        END;
     HUnLock(theDCE^.dCtlStorage);
     IF theDCE^.dCtlRefNum < 0 THEN
        HPurge(Handle(theDCE^.dCtlDriver));
     theDCE^.dCtlWindow := NIL;
     DisposHandle(theDCE^.dCtlStorage);
     theDCE^.dCtlStorage := NIL;
     Close := noErr;
  END;

{  DAs main routine.}
  FUNCTION Main;
  BEGIN
     CASE driveCall OF
        DriverOpen :
           Main := Open(theDCE);
        DriverPrime :
           Main := Prime(theDCE, IOPB);
        DriverControl :
           BEGIN
           BitClr(@theDCE^.dCtlFlags, 15 - dCtlEnable);
           Main := Control(theDCE, IOPB);
           BitSet(@theDCE^.dCtlFlags, 15 - dCtlEnable);
           END;
        DriverStatus :
           Main := Status(theDCE, IOPB);
        DriverClose :
           Main := Close(theDCE, IOPB);
     END;
  END;
END.
```

```
*  ColorImageDA.R
*  Resource Source File
*  by Steve Sheets 12/87
*
ColorImageDA Rsrc

Type MENU
  ,-16000
ColorImage
  About ColorImage...
  (-
  Load Picture
  (-
  Show Picture
  Show Palette

Type STR#
  ,-16000
2
ColorImage
No Picture Available

Type DITL
  ,-16000
5

button
160 170 180 230
OK

staticText
20 95 39 305
ColorImage 1.1 by Steve Sheets

staticText
40 104 59 296
Designed for MacTutor 12/87

staticText
80 40 119 360
DA - Displays a PICT file.  If
it contains a Color PixMap,
displays it in the correct Palette.

staticText
120 40 139 360
FKEY - Saves the current screen
into a PICT file.

Type ALRT
  ,-16000
60 120 260 520
-16000
5555
```

Macintosh II
Graphics Devices & Chroma



Greg Marriott
The MacHax™ Group
MacTutor Vol. 4 No. 3

The Good(?) Old Days

How many of you Mac programmers out there remember how hard it was to master Quickdraw? Come on, raise your hands... Now, how many of you still have trouble sometimes? (Yes, my hand is up, too!) Remember those long hours spent trying to figure out how SetOrigin works, and wondering which regions stick to the window and which ones stick to the GrafPort coordinates? Wasn't it a bummer trying to remember why FrameRgn and FramePoly produce different results? I can see you grimacing as you recall the many hours spent wondering where your pictures go when you offset them. I, for one, refused to knuckle under, and finally managed to get most of it figured out. From the abundance of Mac software out there, it looks like a lot of other people persevered as well.

Apple Consults Ted Turner (or Color, Anyone?)

Just when we had everything under control, along came Color Quickdraw! With it came several new concepts to absorb. The most obvious addition to the Mac's graphics system is, of course, color. Obvious, that is, to all those lucky enough to have a color monitor attached. People with monochrome screens have to settle for lots of shades of gray. **Really** lucky people can have monochrome screens and color screens hooked up at the same time! This is where I (very smoothly) slide into a discussion of another addition to the Quickdraw system: the ability to have several display devices instead of a lone monitor. The phrase "display device" could apply to a wide variety of equipment, but I'll limit my discussion to the CRTs we've come to know and love. (For a description of how the color part of Color Quickdraw works, read Inside Macintosh Vol. 5, and take a look at Steven Sheets' article in the September '87 MacTutor).

Six of One, Half a Dozen of the Other

As most of you know, the Mac II has expansion slots (for you trivia buffs out there, it has six of them). Each of these slots can have something really neat plugged into it, like additional processors, data acquisition boards, and other stuff like that. But, if you don't have a video card and monitor plugged in, you won't be able to see what's going on. I ought to know. My monitor took about a month longer to arrive than my CPU did. The Mac made a nice sound when I turned it on, but that got really boring after the first couple of hundred times. Anyway, with six slots available, enterprising (and loaded) people can plug in six video cards to use with many monitors of various shapes and sizes (some cards allow more than one monitor to be attached).

So, with lots of screens putting quite a load on your table (not to mention your budget), what would you expect to see? Several identical copies of your desktop? Sounds interesting, but not very practical. Quickdraw treats all the video devices collectively as a single, possibly irregular, display. Windows may be placed anywhere on the desktop, and as a result a single window can extend across several monitors. The effect is especially impressive when adjacent monitors have different color environments, or one is color and the other b/w. This spectacular feat is accomplished through careful management of **graphics devices**.

Graphics Devices can be (loosely) described as anything Color Quickdraw can draw into (I'll refer to Color QuickDraw as CQD from here on out). By the way, when I mention a graphics device I'm not talking about the piece of expensive hardware sitting on your desk. I'm referring to the data structure CQD uses to describe the piece of expensive hardware. A **GDevice** record holds information about the device's postition in Quickdraw space, the color capabilities, color matching routines, and cursor handling on the device. Physical devices also have drivers associated with them, identified by the **gdRefNum** field, to handle things like changing the depth or color environment. For multiple screens, CQD maintains a linked list of GDevices (actually, it's a list of handles to GDevices), one for each monitor. The head of the list is in a low memory global called **DeviceList**. The screen with the menu bar is described by the GDevice in **MainDevice**, also in low memory. When drawing is happening on a device, a handle to it is stored in **theGDevice**.

Normally, applications should ignore the fact that their windows might extend over more than one screen and let Quickdraw take care of the details. As a matter of fact, programs should never (well, almost never) depend on any particular device configuration, such as the number of colors present. The user can change the environment any time he wants to, one way being through the Control Panel.

Sometimes, though, taking advantage of the way graphics devices work can be useful. I've written a desk accessory to help illustrate some points to keep in mind when dealing with graphics devices. I'm not going to go over **everything** you should know about graphics devices here, since Inside Macintosh, Vol. 5 covers them very well.

What a Great Name For a Color DA!

Chroma is a desk accessory that displays all the colors (or gray shades) currently available on attached graphics devices. It pays attention to the depth (bits per pixel) of each device when deciding how many colors to draw. Other useful data are also displayed, such as the slot number a device is plugged into, the refNum of the driver, the GDevice flags, and the device's mode. The flags indicate the presence of several conditions, including whether or not the device is the main screen, was initialized from RAM, was initialized from a 'scrn' resource, is the active screen,

has a driver, and is color or b/w. The mode is the number passed to the driver, if there is one, to set the pixel depth, etc.

When the Chroma window lays across more than one device, the piece of the window on a device reflects that device's environment. That is, a color monitor with 256 colors and a b/w monitor with 16 gray shades would cause Chroma to draw part of the window with lots of little color squares and the other part of the window with big gray ones. The rest of the information provided is also pertinent to the device on which it is drawn. By positioning the slot number on the boundary between screens, you'll see part of a different number on each screen. Chroma starts with a rather small window, so you can leave it around to examine your color environment and not cover up other areas of interest. It supports zooming, though, so you can get a better look at the colors of a device. Chroma's window zooms to fill the screen where the zoom box was when you hit it. (To learn about DA programming in general, see some past issues of MacTutor [Ed, how about suggesting some articles here?]).

A Self-Aware(!) Machine

Since Chroma's activities take place in (potentially) color environments, it needs a color window. But, color windows can only be created in the presence of CQD. So, the first thing Chroma does when it's opened is to make sure CQD is present. Some programs check this by looking at a low-memory global called ROM85 and seeing if they are on a Mac II. It's true that CQD only runs on a Mac II, but that may not always be the case (we might see a color MacSE, but then again, that's just a silly rumor... right?). A much more portable way to detect CQD is by calling **SysEnvirons**. The SysEnvirons call fills in a record describing the computer's configuration. SysEnvirons notes the presence of CQD, a floating-point coprocessor, and the system version, among other things. For a detailed explanation of SysEnvirons, see Apple TechNote #129.

If Chroma is opened on a machine without CQD, it puts up a "regular" window instead of a color window, and draws a picture complaining about the lack of color tables to display. Notice that the b/w window is not allowed to zoom. It seems pretty silly to zoom an error message to fill the screen! Chroma keeps a boolean in its storage handle to indicate the presence (or absence) of CQD, so the window updating mechanism knows whether it can use color drawing commands or not.

How Things Work

Now I come to the "guts" of Chroma, its drawing procedures. Chroma draws the contents of its window whenever it receives an update event from the Desk Manager (or from itself, as explained later). The procedure to draw the contents is named, strangely enough, **DrawWindow**. It is called once for each device the Chroma window intersects. It's easy to check for intersections, since part of a GDevice is the rectangle, in global coordinates, which describes the device's position in Quickdraw space. If the content region of the Chroma window intersects any part of the device, the clip region for the window is set to the intersection and DrawWindow is called.

DrawWindow uses the Palette Manager (PM) to handle its color selections for the various devices. When the color window is created, Chroma makes a palette for it using **NewPalette** and installs it using **SetPalette**. Before drawing on each device, DrawWindow copies the color table from it into the palette using **CTab2Palette**. This call resizes the palette, if necessary, to have the same number of entries as the color table. A changed palette isn't really useful until it's activated, so **ActivatePalette** is called. When ActivatePalette is called with the frontmost window as a parameter, the PM attempts to provide a color environment containing as many colors in the palette as it can. To do this, the PM checks the color tables of each device the window intersects. It changes the color tables to achieve a best fit. But, Chroma isn't designed to change color environments, just to show what's already there. Lucky for us, the PM defines a special kind of palette entry that causes no change in the color environment. These "explicit" colors can be specified in our palette by passing **pmExplicit** in the **srcUsage** parameter of CTab2Palette. (Personal note about the PM: I thought it would be hard to get into the spirit of using the PM, but it's not. Everybody writing color programs should use the PM. Using the PM will help to ensure compatibility with MultiFinder, a wide variety of graphics devices, and future system software).

Chroma decides how big to make its color boxes based on two things: the current depth of the device, and the size of the window. The depth determines how many boxes there will be (2 for 1-bit mode, 4 for 2-bit mode, 16 for 4-bit mode, and 256 for 8-bit mode). Actually, Chroma doesn't really know the depth (pixel size) of a GDevice, but goes to the color table of the device's pixMap to find out how many colors are available. You math majors out there probably noticed that with the exception of 1-bit mode, the number of colors available at each depth is a perfect square. So, a square pattern of square boxes makes the most sense for displaying the colors. The Chroma window starts out wider than it is tall, and the zooming code maintains this relationship (even on skinny monitors like Radius FPD). Chroma calculates the box size as the vertical size of the window divided by the square root of the number of colors. Once the box size is calculated, DrawWindow cycles through the pixel values as it draws color boxes, selecting colors using **PmForeColor** with the pixel value as its index parameter.

The data about the device are drawn in the "wider" part of the window on the right side. The strings identifying each piece of data are in resources, so ambitious users can change them to whatever they want. When the window is created, the strings are scanned to find the longest one, and that's how much wider the window is extended. I decided to print the numbers in hexadecimal, because that is what you'd see in a debugger window. The flags and device mode are best read as binary numbers, but hex numbers are shorter and hackeroids (like me) can convert hex to binary in their heads.

Earlier, I mentioned updates coming from the DA itself. This is necessary to handle moving the window across devices of the same depth. No update event is generated by the system if it can get away with just copying the bits from one place to another. This is exactly what happens, as long as no part of the window moves to a device with a different pixel size. Since things like slot

numbers and flags need to be drawn correctly, Chroma needs to know if the window has moved, so it can correctly update the data. The Desk Manager handles moving the window, without ever notifying the DA (unless part of the window requires updating, in which case an update event is eventually passed along). Well, DAs have the option of periodically being given time by the system occasionally (by setting the drvrDelay field and the needTime flag in the driver header). Chroma wakes up every 10th of a second or so and checks to see if the window has been moved since the last update event. If it has, Chroma invalidates the rectangle containing all the info, thereby generating an update event for itself.

Almost Done

WhichDevice takes a point, in global coordinates, and returns a handle to the GDevice containing that point. This function is used to figure out where to zoom the Chroma window. ZoomWindow normally fills the main device (the one with the menubar), so special care must be taken to zoom the window somewhere else. The stdState field of the WState record attached to the window contains the "zoomed" size assumed by the window in response to a ZoomWindow call with the inZoomOut part code. By setting this rectangle, Chroma controls where the window will go.

Speaking of zooming, DA's can't follow the "normal" detection process when it comes to the zoom box. Normally, applications call FindWindow to see which part of which window was hit by a mouseDown event. The inZoomIn and inZoomOut part codes are returned if the zoom box is hit. But, FindWindow returns inSysWindow for DA windows, no matter which part of the window is hit. DA's can't gain anything by calling FindWindow, because they already **know** the click was somewhere in a DA window! But, with a small bit of trickery, FindWindow can be fooled into thinking the DA window is an application window. The windowKind field of the window is set to the refNum of the DA when the window is opened. The refnum is always negative, and that's how the window manager knows the window belongs to a DA. By temporarily making the windowKind positive, FindWindow returns meaningful part codes instead of inSysWindow. Once a good partCode is returned, the windowKind is made negative again.

Now for another (sort of) tricky part. Once the window is zoomed to fill a monitor, and then moved somewhere else, there's no way to make the window small again. Hitting the zoom box makes the window fill whatever device it's on. So, what I did was add an escape hatch. Holding the **option** key when hitting the zoom box will shrink the window back to its original size. If it's already small, nothing happens.

The slot number of a given device is determined by the **WhereIsTheDevice** function. It takes the refNum of the driver associated with the GDevice, turns it into a unit number, looks up the device control entry in the unit table, and gets the slot number from the DCE. All this assumes the GDevice record belongs to a physical device. If the refNum is zero, indicating no driver, then the function returns an impossible slot number.

Conclusion

Don't you hate those wind-ups where authors thank everybody they've ever known? Well, I don't. I couldn't have done this all by myself, and I want to make sure the people who helped get the credit they deserve. Here they are: First my partner Scott T. Boyd, the best spell checker I've ever used, for being a good sounding board. Next, Chris Derossi, Tech Support Technoid, for his good advice and unending patience with my last minute phone calls. Darin Adler, for some excellent suggestions and for being the best darned Cheese Host I've ever had the pleasure to meet. Now all the rest: TECHNOSTUD, Art, Howard, Doug, John, Thug, Pooter, Spaz, Ronnie, Paul, Ollie, and Bill Casey (who's not really dead, but running the secret government).

Any questions, give me a call or drop me a line.
The MacHax™ Group
3420D Sandra St.
Bryan, TX 77801
(409)846-4102
attn: G. Marriott, M/S 27-AQ
AppleLink: D0635
MCImail: Greg Marriott
BITNET: max@tamlsr

File Chroma.p

```
{
This is a DA to show the current color assignments of
all connected displays, without taking up the
whole screen (unless you want to zoom the window!).

By Greg Marriott

Special thanks to Scott T. Boyd, Chris Derossi,
& Darin Adler

Copyright 1988, The MacHax™ Group.
All rights reserved.

1.0d1   1-23-88     Created DA that puts up a window, and
                    plots colors of MainDevice
1.0d2   1-24-88     Made window wider, added slot number
                    of MainDevice
        1-24-88     Beefed up error detection(but not
                    necessarily correction %-)
1.0d3   1-25-88     Added brag pictures, b/w pictures
        1-25-88     Call SysEnvirons to detect colorQD
        1-25-88     Added word 'Slot' above slot number
1.0d4   1-26-88     Miscellaneous bug/cosmetic fixes
1.0d5   1-28-88     Made updates smarter. Now it clips to
                    each device and draws with their
                    characteristics(i.e. depth and color
                    table)
        1-28-88     Added zooming, then fixed zooming
        1-29-88     Revised zooming to fill device of
                    interest, instead of main device
1.0b1   1-29-88     Revved to beta and selected testers
1.0b2   2- 4-88     Scott made zooming work right,allowing
                    the use of FindWindow.
1.0b3   2- 7-88     Began to use Palette Manager, as Chris
                    suggested.
        2- 8-88     Added more slot info to window,
                    including gdFlags, gdRefNum, and
                    gdMode, all hex numbers.
        2- 8-88     Revised zooming, added option
                    zoom feature
1.0     2- 9-88     Published!
}
```

```
{$D+} { Debugging symbols on }

UNIT Chroma;

INTERFACE

USES
  MemTypes, QuickDraw, OSIntf, ToolIntf, PaletteMgr,
  PackIntf;


FUNCTION DRVROpen(ctlPB:ParmBlkPtr;
                  dCtl:DCtlPtr):OSErr;
FUNCTION DRVRControl(ctlPB:ParmBlkPtr;
                     dCtl:DCtlPtr):OSErr;
FUNCTION DRVRStatus(ctlPB:ParmBlkPtr;
                    dCtl:DCtlPtr):OSErr;
FUNCTION DRVRPrime(ctlPB:ParmBlkPtr;
                   dCtl:DCtlPtr):OSErr;
FUNCTION DRVRClose(ctlPB:ParmBlkPtr;
                   dCtl:DCtlPtr):OSErr;


IMPLEMENTATION

CONST
  versID = '1.0';
  windowVSize = 96;

  bwChromaPicture = 0;
  colorChromaPicture = 1;
  sorryPicture = 2;

  infoStrings = 0;
    firstInfoStr = 1;
    slotStr = 1;
    refNumStr = 2;
    flagsStr = 3;
    modeStr = 4;
    lastInfoStr = 4;

TYPE
  ChromaStorage = RECORD
    theDevice:GDHandle;
    minSize:Point;
    windowPlace:Rect;
    hasColorQD:Boolean;
  END;
  ChromaStoragePtr = ^ChromaStorage;
  ChromaStorageHandle = ^ChromaStoragePtr;

  EventPtr = ^EventRecord;
  trix =  RECORD  { needed for some coercive behavior }
    CASE Boolean OF
      TRUE : (dontCare:ARRAY[0..10] OF Integer);
      FALSE : (theEventPtr:  EventPtr);
    END;

  WStatePtr = ^WStateData;
  WStateHandle = ^WStatePtr;

FUNCTION RsrcID(dCtl:DCtlPtr;localID:INTEGER):Integer;
BEGIN
    { Calculate resource ID's based on our refNum }
  RsrcID := (BOR($C000,(BSL(BNOT(dCtl^.dCtlRefNum),5))))
            + localID;
END;

FUNCTION DRVROpen(ctlPB:ParmBlkPtr;dCtl:DCtlPtr):OSErr;

  FUNCTION MinWindowSize:Point;
  VAR
    theStringList:Handle;
    theString:Str255;
    widest:INTEGER;
    i:INTEGER;
  BEGIN
      { The vertical minimum is predetermined }
    MinWindowSize.v := windowVSize;
    widest := 0;

      { Read all the strings, finding the widest one }
    FOR i := firstInfoStr TO lastInfoStr DO
    BEGIN
      GetIndString(theString, RsrcID(dCtl, infoStrings),
                   i);
      IF StringWidth(theString) > widest THEN
        widest := StringWidth(theString);
    END;

      { Add a little margin }
    widest := widest + 2;
    MinWindowSize.h := windowVSize + widest;
  END; { FUNCTION MinWindowSize }

VAR
  oldPort:GrafPtr;
  wFrame:Rect;
  myWindow:WindowPtr;
  myWindowSize:Point;
  theEnv:SysEnvRec;
  envError:OSErr;
  aPalette:PaletteHandle;

BEGIN
    { If there's no window yet, put one up... }
  IF dCtl^.dCtlWindow = NIL THEN
  BEGIN
      { Pick a size for the window, even though we're
        gonna change that before the window
        becomes visible }
    SetRect(wFrame, 0, 0, windowVSize, windowVSize);
    OffSetRect(wFrame, 100, 100);

      { Ask SysEnvirons about colorQD. If no, put up
        a "regular" window instead of a color window }
    envError := SysEnvirons(1, theEnv);
    IF theEnv.hasColorQD THEN
      myWindow := NewCWindow(NIL, wFrame, 'Chroma',
                    FALSE, zoomNoGrow, WindowPtr(-1),
                    TRUE, LongInt(0))
    ELSE
      myWindow := NewWindow(NIL, wFrame, 'Chroma',
                    FALSE, noGrowDocProc,
                    WindowPtr(-1), TRUE, LongInt(0));

      { If the window was created OK, figure out how big
        the window should be and then fill in some
        important fields }
    IF myWindow <> NIL THEN
    BEGIN
        { Start by setting the text parameters
          and computing a minimum size for the window }
      SetPort(myWindow);
      TextMode(SrcOr);
      TextFont(Geneva);
      TextSize(9);
      TextFace([]);
      myWindowSize := MinWindowSize;

        { Set the window to the min. size, and show it }
      SizeWindow(myWindow, myWindowSize.h,
                 myWindowSize.v, FALSE);
      ShowWindow(myWindow);
      SelectWindow(myWindow);

        { This one shows that a DA owns the window }
```

```
      WindowPeek(myWindow)^.WindowKind :=
        dCtl^.dCtlRefNum;

        { And this one lets the desk manager know, too }
      dCtl^.dCtlWindow := Ptr(myWindow);

        { We'll keep some info in the storage field }
      dCtl^.dCtlStorage := NewHandle(SizeOf(
                             ChromaStorage));

        { Note whether we put up a color window or not }
      ChromaStorageHandle(dCtl^.dCtlStorage)^^.
        hasColorQD := theEnv.hasColorQD;

        { Store the min. size, based on the longest
          description name in the string list }
      ChromaStorageHandle(dCtl^.dCtlStorage)^^.
        minSize := myWindowSize;

      IF theEnv.hasColorQD THEN
      BEGIN
          { Get the main device, then note the window's
            global location }
        WITH ChromaStorageHandle(dCtl^.dCtlStorage)^^ DO
        BEGIN
          theDevice := GetMainDevice;
          windowPlace := WindowPeek(dCtl^.dCtlWindow)^.
                           contRgn^^.rgnBBox;
        END;

          { Make a new palette, and "install" it }
        aPalette := NewPalette(256, NIL, pmExplicit, 0);
        SetPalette(myWindow, aPalette, TRUE);
      END;
    END;{ if window was created ok... }
  END;{ if window doesn't exist... }
  DRVROpen := NOErr;
END;

FUNCTION DRVRClose(ctlPB:ParmBlkPtr;dCtl:DCtlPtr):OSErr;
BEGIN
    { If there's a window up, get rid of it,
      and let the desk manager know, too }
  IF dCtl^.dCtlwindow <> NIL THEN
  BEGIN
    DisposeWindow(WindowPtr(dCtl^.dCtlWindow));
    dCtl^.dCtlWindow := NIL;
  END;

    { If we created storage, get rid of it }
  IF dCtl^.dCtlStorage <> NIL THEN
    DisposHandle(dCtl^.dCtlStorage);

  DRVRClose := NOErr;
END;

FUNCTION DRVRControl(ctlPB:ParmBlkPtr;
                     dCtl:DCtlPtr):OSErr;

  FUNCTION WhereIsTheDevice(whatDevice:GDHandle):
           INTEGER;
  CONST
    UTableBase = $11c;
  VAR
    devRefNum:Longint;
    unitTablePtr:^Ptr;
    unitTableBase:Ptr;
    theDCEHandle:AuxDCEHandle;
    theDCEHandlePtr:^AuxDCEHandle;
  BEGIN
      { The refnum of the gdevice we're interested in }
    devRefNum := whatDevice^^.gdRefNum;

      { If it's zero, return an impossible slot number }
    IF devRefNum = 0 THEN
    BEGIN
      WhereIsTheDevice := 42;
      Exit(WhereIsTheDevice);
    END;

      { Turn it into a unit number }
    devRefNum := BNOT(devRefNum);

      { Grab the unit table address }
    unitTablePtr := Pointer(UTableBase);
    unitTableBase := unitTablePtr^;

      { Go get the address of the device control entry
        for this unit }
    theDCEHandlePtr := Pointer(Ord4(unitTableBase)
                         + devRefNum * 4);
    theDCEHandle := theDCEHandlePtr^;

      { Return the slot number of this driver }
    WhereIsTheDevice := theDCEHandle^^.dCtlSlot;
  END;{ FUNCTION WhereIsTheDevice }

  PROCEDURE CenterInWindow(VAR theRect:Rect);
  VAR
    rectSize:Point;
    chromaSize:Point;
  BEGIN
      { "Home" the rect first }
    OffsetRect(theRect, -theRect.left, -theRect.top);

      { Figure size of rect }
    WITH theRect DO
      SetPt(rectSize, right, bottom);

      { Figure size of window }
    WITH GrafPtr(dCtl^.dCtlWindow)^.portRect DO
      SetPt(chromaSize, right - left, bottom - top);

      { Position the rect, centered in the window }
    OffsetRect(theRect, (chromaSize.h - rectSize.h)
                DIV 2, (chromaSize.v - rectSize.v)
                DIV 2);
  END;{ PROCEDURE CenterInWindow }

  PROCEDURE DrawWindow;

    PROCEDURE DrawNextString(VAR drawWhere:Point;
                             drawWhat:Str255;
                             drawHow:Style);
    VAR
      someFontInfo:FontInfo;
      oldFace:Style;
    BEGIN
        { Save the text style }
      oldFace := GrafPtr(dCtl^.dCtlWindow)^.txFace;
        { Set the desired text face }
      TextFace(drawHow);

        { The "cursor" will be centered already, so move
          back half the width and draw it. }
      Move(-(StringWidth(drawWhat) DIV 2), 0);
      DrawString(drawWhat);

        { Move the "cursor" up a line, since
          we're drawing from the bottom of the window
          toward the top. We're letting the descenders
          overhang, since textmode has been set to SrcOr
          and won't wipe out the overlap. }
      GetFontInfo(someFontInfo);
      drawWhere.v := drawWhere.v - someFontInfo.ascent;
        { Re-center the "cursor" for the next time }
      MoveTo( drawWhere.h, drawWhere.v );
```

```
    { Restore text face }
  TextFace(oldFace);
END;{ PROCEDURE DrawNextString }

PROCEDURE NumToHexString(aNumber:INTEGER;
                         VAR aString:Str255);
  { Make a 4 digit hexadecimal number
    from an integer }
VAR
  unsignedNum:LONGINT;
  i:INTEGER;
  aDigit:INTEGER;
BEGIN
    { Make an unsigned hex number from the
      signed integer }
  IF aNumber>0 THEN
    unsignedNum := aNumber
  ELSE
    unsignedNum := aNumber + 65536;

    { Make sure the string has 4 digits (spaces)}
  aString := '    ';

    { Fill in the digits, from lsd to msd }
  FOR i := 4 DOWNTO 1 DO
  BEGIN
    aDigit := unsignedNum MOD 16;
    IF aDigit<10 THEN
      aString[i] := chr(ord('0')+aDigit)
    ELSE
      aString[i] := chr(ord('A')+aDigit-10);
    unsignedNum := unsignedNum DIV 16;
  END;
END;{ PROCEDURE NumToHexString }

VAR
  howManyColors:INTEGER;
  pixPerSide:Point;
  pixSize:Point;
  portSize:Point;
  pixNumber:INTEGER;
  oldIndex:INTEGER;
  i,j:INTEGER;
  slotNumberString:Str255;
  theFontInfo:FontInfo;
  sorryPict:Handle;
  sorryRect:Rect;
  aPalette:PaletteHandle;
  cursorPt:Point;
  theString:Str255;
BEGIN
  IF dCtl^.dCtlWindow <> NIL THEN
  BEGIN
    WITH ChromaStorageHandle(dCtl^.dCtlStorage)^^ DO
    BEGIN
        { Prepare to draw the contents of the window }
      EraseRect(GrafPtr(dCtl^.dCtlWindow)^.portRect);

        { On a color Mac with CLUT device, show the
          colors, on other machines, a warning }
      IF hasColorQD AND (theDevice^^.gdType=clutType)
      THEN
        BEGIN
            { What is the size of our window? }
          WITH GrafPtr(dCtl^.dCtlWindow)^.portRect DO
            SetPt(portSize, right - left,
                            bottom - top);

            { We need to know how many colors are in
              the clut of the device of interest }
          howManyColors := theDevice^^.gdPMap^^
                           .pmTable^^.ctSize + 1;

            { Assume a square arrangement of colors,
              except for 2 color mode. Calculate pixel
              size based on vertical size of window,
              assuming that the window will always
              be wider than it is tall }
          pixPerSide.v := Round(sqrt(howManyColors));
          IF howManyColors = 2 THEN
            pixPerSide.h := 2
          ELSE
            pixPerSide.h := pixPerSide.v;

          pixSize.v := portSize.v DIV pixPerSide.v;
          IF howManyColors = 2 THEN
            pixSize.h := pixSize.v DIV 2
          ELSE
            pixSize.h := pixSize.v;

          PenSize(pixSize.h, pixSize.v);

            { Keep track of the pixel number and use
              it to plot the colors in the palette
              we'll fill from the device's CLUT }
          pixNumber := 0;

            { Fill the palette with the colors from
              the current device, and activate the
              changes }
          aPalette := GetPalette(WindowPtr(dCtl^
                                 .dCtlWindow));
          CTab2Palette(theDevice^^.gdPMap^^.pmTable,
                       aPalette, pmExplicit,0);
          ActivatePalette(WindowPtr(dCtl^
                          .dCtlWindow));

            { Plot colored pixels one row at a time,
              'til they're all used up }
          FOR i := 0 TO(pixPerSide.v - 1) DO
            FOR j := 0 TO(pixPerSide.h - 1) DO
            BEGIN
                { Set the foreColor, and draw a dot }
              PmForeColor(pixNumber);
              MoveTo(j * pixSize.h, i * pixSize.v);
              Line(0, 0);

              pixNumber := pixNumber + 1;
            END;{ Plotting loop... }

            { Put up some device info, first figuring
              some window positions to put the text,
              starting at the bottom of the window and
              centered in the blank part of the
              window }
          SetPt(cursorPt, portSize.v + (portSize.h -
                portSize.v) DIV 2, portSize.v - 2 );
          MoveTo( cursorPt.h, cursorPt.v );

            { Put up the mode, then its rsrc string }
          NumToHexString(theDevice^^.gdMode,
                         theString);
          DrawNextString(cursorPt, theString, [bold]);
          GetIndString(theString, RsrcID(dCtl,
                       infoStrings), modeStr);
          DrawNextString(cursorPt, theString, []);

            { Put up flags, then their rsrc string }
          NumToHexString(theDevice^^.gdFlags,
                         theString);
          DrawNextString(cursorPt, theString, [bold]);
          GetIndString(theString, RsrcID(dCtl,
                       infoStrings), flagsStr);
          DrawNextString(cursorPt, theString, []);

            { Put up refNum, then its rsrc string }
          NumToHexString(theDevice^^.gdRefNum,
                         theString);
```

```
        DrawNextString(cursorPt, theString, [bold]);
        GetIndString(theString, RsrcID(dCtl,
                     infoStrings), refNumStr);
        DrawNextString(cursorPt, theString, []);

          { Put up the slot#, then its rsrc string }
        NumToHexString(WhereIsTheDevice(theDevice) -
                       8, theString);
        DrawNextString(cursorPt, theString, [bold]);
        GetIndString(theString, RsrcID(dCtl,
                     infoStrings), slotStr);
        DrawNextString(cursorPt, theString, []);

      END { if on a mac II... }
    ELSE
      BEGIN
          { For non-color Macs, or non-clut devices,
            show a picture saying why Chroma isn't
            showing pretty colors... }
        sorryPict := GetResource('PICT',
                     RsrcID(dCtl, sorryPicture));

        IF sorryPict <> NIL THEN
        BEGIN
            { Center the picture in the window,
              and draw it }
          sorryRect := PicHandle(sorryPict)^^
                       .picFrame;
          CenterInWindow(sorryRect);

          DrawPicture(PicHandle(sorryPict),
                      sorryRect);
          ReleaseResource(sorryPict);
        END;{ if picHandle is good... }
      END;{ non-color/non-clut condition }

      { Put up the version number in the
        top right corner}
    GetFontInfo(theFontInfo);
    MoveTo(GrafPtr(dCtl^.dCtlWindow)^.portRect.
           right - StringWidth(versID), theFontInfo.
           ascent);
    DrawString(versID);
  END; { WITH ChromaStorage... }
END;{ if window exists... }
END;{ PROCEDURE DrawWindow }

FUNCTION WhichDevice(aPoint:Point):GDHandle;
VAR
  aDevice:GDHandle;
  foundOne:Boolean;
BEGIN
    { Get first one, and set the initial condition }
  aDevice := GetDeviceList;
  foundOne := FALSE;

  WHILE (aDevice <> NIL) AND NOT foundOne DO
    { assuming it has to be in one of the devices... }
  BEGIN
      { Check to see if this is the one... }
    IF PtInRect(aPoint, aDevice^^.gdRect) THEN
    BEGIN
      WhichDevice := aDevice;
      foundOne := TRUE;
    END;
      { Get the next device in the list }
    aDevice := aDevice^^.gdNextGD;
  END;
END;{ FUNCTION WhichDevice }

FUNCTION EqualRectSize(rect1, rect2:Rect):Boolean;
  { This function is dedicated to Scott T. Boyd %-)}
BEGIN
  EqualRectSize :=((rect1.right - rect1.left) =
                   (rect2.right - rect2.left)) AND
                   ((rect1.bottom - rect1.top) =
                   (rect2.bottom - rect2.top));
END;{ FUNCTION EqualRectSize }

PROCEDURE ZoomIt(partCode:INTEGER; clickedWhere:Point;
                 optKeyDown:Boolean);
CONST
  mBarHeight = $BAA;
VAR
  oldRect, newRect:Rect;
  maxHeight:INTEGER;
  menuBarHeight:^INTEGER;
  doZoom:Boolean;
BEGIN
    { Assume we'll zoom }
  doZoom := TRUE;
    { Get the old window size, described by the
      content region }
  oldRect := WindowPeek(dCtl^.dCtlWindow)^.
             contRgn^^.rgnBBox;

    { If the optionKey is down, shrink the window to
      original size... unless, of course, it's already
      original size. }
  IF optKeyDown THEN BEGIN
    newRect := oldRect;
    newRect.left := newRect.right -
                    ChromaStorageHandle(dCtl^.
                    dCtlStorage)^^.minSize.h;
    newRect.bottom := newRect.top +
                      ChromaStorageHandle(dCtl^.
                      dCtlStorage)^^.minSize.v;

      { If it's not already small, make it small,
        otherwise DON'T ZOOM THE WINDOW! }
    IF NOT EqualRect(oldRect,newRect) THEN
      WITH WindowPeek(dCtl^.dCtlWindow)^ DO
        wStateHandle(dataHandle)^^.stdState := newRect
    ELSE
      doZoom := FALSE;
  END
  ELSE BEGIN
      { Figure out which device the zoom box was on,
        so we know which monitor to fill up with
        the window. }
    newRect := WhichDevice(clickedWhere)^^.gdRect;
    InsetRect(newRect, 3, 3);
    newRect.top := newRect.top + 18;{ titlebar }
      { If it's the main device, make room for
        menu bar }
    IF WhichDevice(clickedWhere)=GetMainDevice THEN
    BEGIN
      menuBarHeight := Pointer( mBarHeight );
      newRect.top := newRect.top + menuBarHeight^;
    END;

      { Make sure that the window will be wider than
        it is tall. It should be no taller than its
        width minus the slot info area on the right
        side.  The info area's size can be calculated
        from the minSize width - minSize height. }
    WITH ChromaStorageHandle(dCtl^.dCtlStorage)^^ DO
      maxHeight := newRect.right - newRect.left -
                   (minSize.h - minSize.v);
      { If it's higher than it should be, shorten it }
    IF newRect.bottom - newRect.top > maxHeight THEN
      newRect.bottom := newRect.top + maxHeight;

      { If we're not already a big window, zoom to it,
        by putting the new size in stdState.
        Otherwise, leave the old size there
        (from last time)}
    IF NOT EqualRect(oldRect, newRect) THEN
```

```
      WITH WindowPeek(dCtl^.dCtlWindow)^ DO
        wStateHandle(dataHandle)^^.stdState :=
          newRect;
  END;{ if not optionkey... }

  IF doZoom THEN
  BEGIN
    EraseRect(GrafPtr(dCtl^.dCtlWindow)^.portRect);
    ZoomWindow(WindowPtr(dCtl^.dCtlWindow), partCode,
               false);
    InvalRect(GrafPtr(dCtl^.dCtlWindow)^.portRect);

      { Fool ZoomWindow into always zooming "out" by
        changing the "zoomed" window size }
    WITH WindowPeek(dCtl^.dCtlWindow)^ DO
      wStateHandle(dataHandle)^^.stdState := oldRect;
  END;
END;{ PROCEDURE ZoomIt }

PROCEDURE BragALittle;
VAR
  aboutPict:Handle;
  aboutRect:Rect;
BEGIN
  WITH ChromaStorageHandle(dCtl^.dCtlStorage)^^ DO
  BEGIN
    EraseRect(GrafPtr(dCtl^.dCtlWindow)^.portRect);

      { If they're not on a color Mac, or there's less
        than 16 colors, put up the monochrome version
        of the brag picture.  BTW, the boolean short-
        circuits on non-color Macs so Chroma won't
        try to access non-existent data structures }
    IF (NOT hasColorQD) |
        (theDevice^^.gdPMap^^.pmTable^^.ctSize + 1 <
          16)
    THEN
      aboutPict := GetResource('PICT',
                    RsrcID(dCtl, bwChromaPicture))
    ELSE
      aboutPict := GetResource('PICT',
                    RsrcID(dCtl,
                           colorChromaPicture));

    IF aboutPict <> NIL THEN
    BEGIN
        { Center the picture, and draw it }
      aboutRect := PicHandle(aboutPict)^^.picFrame;
      CenterInWindow(aboutRect);

      DrawPicture(PicHandle(aboutPict), aboutRect);
      ReleaseResource(aboutPict);
    END;

      { Wait until the button's not down
        (Hi Roger %-) }
    REPEAT UNTIL NOT StillDown;

      { Restore the contents of the window }
    InvalRect(GrafPtr(dCtl^.dCtlWindow)^.portRect);
  END;{ WITH ... }
END;{ PROCEDURE BragALittle }

VAR
  eventAt:  EventPtr;
  oldPort:GrafPtr;
  localPoint:Point;
  aDevice:GDHandle;
  clippyPart:Rect;
  oldClip:RgnHandle;
  thePartCode:INTEGER;
  whichWindow:WindowPtr;
  intersections:INTEGER;
  dumbRect:Rect;

BEGIN { of DRVRControl }
  WITH ChromaStorageHandle(dCtl^.dCtlStorage)^^ DO
  BEGIN
    GetPort(oldPort);
    SetPort(GrafPtr(dCtl^.dCtlWindow));

    CASE ctlPB^.csCode OF
      accEvent:
        BEGIN
            { Get the event pointer, through some
              coercive trickery }
          eventAt := trix(ctlPB^.CSParam).theEventPtr;
          WITH eventAt^ do
          BEGIN
            CASE what OF
              updateEvt:
                BEGIN
                    { Save the current clip region, and
                      restore it after the update }
                  oldClip := NewRgn;
                  GetClip(oldClip);
                  BeginUpdate(WindowPtr(eventAt^.
                              message));
                    { If there's no ColorQD, just draw
                      the  window; otherwise cycle
                      through all the devices, clipping
                      the window to each device }
                  IF NOT hasColorQD THEN
                    DrawWindow
                  ELSE BEGIN
                    aDevice := GetDeviceList;
                      { Go until the end of the list }
                    WHILE aDevice <> NIL DO
                    BEGIN
                        { Check for intersection of the
                          content rgn with the device }
                      WITH WindowPeek(dCtl^.dCtlWindow)^
                      DO IF SectRect(contRgn^^.rgnBBox,
                                     aDevice^^.gdRect,
                                     clippyPart) THEN
                      BEGIN
                          { Translate the global rect
                            to local coordinates }
                        GlobalToLocal(clippyPart.
                                      topLeft);
                        GlobalToLocal(clippyPart.
                                      botRight);

                          { Set the clip, so other parts
                            don't get blasted.
                            Set theDevice field of the
                            storageHandle to the device
                            of interest, then draw }
                        ClipRect(clippyPart);
                        theDevice := aDevice;
                        DrawWindow;
                      END;{ if intersect... }
                      { Go to the next device in the
                        device list }
                    aDevice := aDevice^^.gdNextGD;
                    END;{ while there's more devices...}
                  END;{ if a color machine... }

                  EndUpdate(WindowPtr(eventAt^.
                            message));

                    { Restore the clip, axe the region }
                  SetClip(oldClip);
                  DisposeRgn(oldClip);

                    { Mark the window's position }
                  windowPlace := WindowPeek(dCtl^.
                       dCtlWindow)^.contRgn^^.rgnBBox;
```


© The Definitive MacTutor, Vol. 4

```
            END;
          mouseDown:
            BEGIN
              localPoint := where;
              GlobalToLocal(localPoint);

              { Pretend this isn't a DA window to
                fool FindWindow }
              WITH WindowPeek(dCtl^.dCtlWindow)^ DO
                windowKind := -windowKind;
              thePartCode := FindWindow(where,
                             WindowPtr(dCtl^.
                             dCtlWindow));
              WITH WindowPeek(dCtl^.dCtlWindow)^ DO
                windowKind := -windowKind;

                { In zoom box, do it, passing along
                  the state of the option key... }
              IF (thePartCode=inZoomIn) OR
                 (thePartCode=inZoomOut) THEN
              BEGIN
                IF TrackBox(WindowPtr(dCtl^
                            .dCtlWindow), where,
                            thePartCode) THEN
                  ZoomIt(thePartCode, where,
                         BitAnd(modifiers,
                                optionKey)<>0);
              END ELSE
                  { See if they clicked on the vers.
                    number.  If so, put up a brag
                    picture 'til they release the
                    button }
                WITH localPoint,
                    GrafPtr(dCtl^.dCtlWindow)^.
                      portRect DO
                  IF (h > right-StringWidth(versID))
                      AND (v < 12) THEN
                  BEGIN
                    BragALittle;
                  END;{ if h and v in range... }
            END;{ mouseDown case }
        END;{ case what ... }
      END;{ with eventAt... }
    END;{ accEvent Case }

  accRun:
    BEGIN      { the periodic call }
      WITH ChromaStorageHandle(dCtl^.
           dCtlStorage)^^ DO
      BEGIN
          { This is a problem on multiple device
            systems, check for CQD }
        IF hasColorQD THEN
        BEGIN
            { Moved? If yes, inval the info part
              of the window }
          IF NOT EqualRect(WindowPeek(dCtl^.
                           dCtlWindow)^.
                           contRgn^^.rgnBBox,
                           windowPlace) THEN
          BEGIN
            clippyPart :=GrafPtr(dCtl^.dCtlWindow)^.
                                 portRect;
            clippyPart.left := clippyPart.left +
                               clippyPart.bottom;
            InvalRect(clippyPart);
          END;{ if window moved... }
        END;{ if has color quickdraw... }
      END;{ with chromastorage... }
    END;{ accRun case... }
END;{ case ctlPB^.csCode }

SetPort(oldPort);
DRVRControl := NoErr;
  END;{ with dctl^.dctlstorage... }
END;{ of DRVRControl }

FUNCTION DRVRPrime(ctlPB:ParmBlkPtr;dCtl:DCtlPtr):OSErr;
BEGIN
  DRVRPrime := NoErr;
END;

FUNCTION DRVRStatus(ctlPB:ParmBlkPtr;dCtl:DCtlPtr):OSErr;
BEGIN
  DRVRStatus := NoErr;
END;
END. {of Chroma UNIT}
```

Chroma Make File for MPW

```
Chroma.DA  ƒ  Chroma.DRVW Chroma.r
  Rez -rd -c DMOV -t DFIL Chroma.r -o Chroma.DA
Chroma.DRVW  ƒ  Chroma.p.o
  Link -w -rt DRVW=0 ∂
    -sg Chroma ∂
    "(Libraries)"DRVRRuntime.o ∂
    Chroma.p.o ∂
    "(Libraries)"Interface.o ∂
    "(PLibraries)"Paslib.o ∂
    -o Chroma.DRVW -c "????" -t "????"
Chroma.p.o  ƒ  Chroma.p
  Pascal Chroma.p
```

Chroma Resource File In REZ Format

```
/*
 * File Chroma.r
 *
 * Copyright The MacHax™ Group, 1988
 * All rights reserved.
 *
 */
#include "Types.r"          /* To get system types */
#include "MPWTypes.r"       /* To get 'DRVW' type */

type 'DRVR' as 'DRVW';      /* Map 'DRVW' => 'DRVR' */

/*
 * This will produce a DRVR resource from the special
 * DRVW type.
 *
 * Note that the ID 12 is irrelevant, since the
 * Font/DA Mover will renumber it
 * to something else when installing it anyway.
 *
 * The leading NUL in the resource name is required to
 * conform to the desk accessory naming convention.
 *
 * The resource is declared purgeable.  If the code were
 * to do funky things like SetTrapAddress calls
 * (requiring the code to be around at all times), we
 * would have to set it nonpurgeable.
 */

#define DriverID  12
resource 'DRVR' (DriverID, "\0x00Chroma", purgeable) {
  /*
   * DRVR flags
   */
  dontNeedLock,             /* OK to float around */
  needTime,                 /* Periodic Control calls */
  dontNeedGoodbye,          /* No special requirements */
  noStatusEnable,
  ctlEnable,                /* Enable Control calls */
  noWriteEnable,
  noReadEnable,
  6,                        /* drvrDelay - 1/10 second */
  updateMask + mDownMask,   /* drvrEMask - Event mask */
```

```
  0,                        /* drvrMenu - No menu */
  "Chroma",                 /* drvrName */
  /*
   * This directive inserts the contents of the
   * DRVW resource produced by linking DRVRRuntime.o
   * with our DA code
   */
  $$resource("Chroma.DRVW", 'DRVW', 0)
};

#define infoStrings 0
resource 'STR#' ( ( 0xC000 | ( DriverID << 5 ) ) +
                  infoStrings ) {
  {
    "Slot";
    "gdRefNum";
    "gdFlags";
    "gdMode"
  };
};

#define bwChromaPicture 0
#define colorChromaPicture 1
#define sorryPicture 2
resource 'PICT' ( ( 0xC000 | ( DriverID << 5 ) ) +
                  colorChromaPicture ) {
  2504,
  (13, 137, 96, 265),
  $"0011 02FF 0C00 FFFF FFFF 0089 0000 000D"
  $"0000 0109 0000 0060 0000 0000 0000 001E"
  $"0001 000A 0000 0000 015E 01B8 0098 8040"
  $"000D 0089 0060 0109 0000 0000 0000 0000"
  $"0048 0000 0048 0000 0000 0004 0001 0004"
  $"0000 0000 0000 1F10 0000 0000 0000 0004"
  $"8000 000F 0000 FFFF FFFF FFFF 0000 FC00"
  $"F37D 052F 0000 FFFF 648A 028C 0000 DD6B"
  $"08C2 06A2 0000 F2D7 0856 84EC 0000 46E3"
  $"0000 A53E 0000 0000 0000 D400 0000 0241"
  $"AB54 EAFF 0000 1F21 B793 1431 0000 0000"
  $"64AF 11B0 0000 5600 2C9D 0524 0000 90D7"
  $"7160 3A34 0000 C000 C000 C000 0000 8000"
  $"8000 8000 0000 4000 4000 4000 0000 0000"
  $"0000 0000 000D 0089 0060 0109 000D 0089"
  $"0060 0109 0000 02C1 0002 C100 02C1 0008"
  $"F000 0201 1110 D500 0EF7 0000 03FE 33FE"
  $"0002 1111 10D5 000F F800 0003 FD33 0530"
  $"0000 1111 10D5 000B F800 FB33 0300 0111"
  $"11D4 0010 F900 0003 FE33 0603 3333 0001"
  $"1111 D400 21F9 00FE 3307 0000 3333 0001"
  $"1110 FC00 0188 88FA 0003 0777 7770 F100"
  $"FE44 0040 FA00 26F9 0008 3333 3000 0333"
  $"3300 01FD 1102 1000 00FC 8803 8000 0007"
  $"FD77 0100 00FC 66F9 00FB 44FB 0029 FA00"
  $"0803 3333 0000 0333 3300 FB11 0100 00FB"
  $"8801 0000 FC77 0170 00FC 6600 60FE 66FE"
  $"0000 04FB 44FB 002B FA00 0803 3333 0000"
  $"0333 3000 FB11 0210 0008 FC88 0100 07FC"
  $"7701 7000 F866 0260 0000 FE44 0340 0444"
  $"44FB 002E FA00 0203 3330 FB00 FB11 0810"
  $"0008 8888 8088 8800 FE77 0400 0777 7000"
  $"FE66 0006 FB66 0800 0444 4440 0004 4444"
  $"FB00 30FA 0002 0333 30FB 00FE 1107 1001"
  $"1110 0008 8888 FD00 0B77 7770 0007 7770"
  $"0066 6660 00FB 6608 0004 4444 0000 0444"
  $"40FB 0034 FA00 0203 3330 FB00 FE11 0710"
  $"0111 1000 8888 80FE 000C 0777 7700 0007"
  $"7770 0066 6660 00FE 660B 0066 6600 4444"
  $"4000 0044 4440 FB00 34FA 0002 0333 30FB"
  $"00FE 1107 0001 1110 0088 8880 FE00 1B07"
  $"7777 0000 0777 7006 6666 0000 6666 6000"
  $"6666 0044 4440 0000 4444 40FB 0032 FA00"
  $"0203 3330 FB00 FE11 0600 0111 1000 8888"
  $"FD00 1A07 7770 0000 0777 7006 6666 0006"
  $"6666 0000 6666 0044 4400 0004 4444 FA00"
  $"32FA 0002 0333 30FB 00FE 1106 0011 1110"
  $"0088 88FD 001A 0777 7000 0077 7770 0666"
  $"6000 0666 6600 0066 6600 4444 0000 0444"
  $"44FA 0032 FA00 0203 3333 FB00 0911 1110"
  $"0011 1110 0888 88FD 0017 0777 7000 0077"
  $"7770 0666 6000 0666 6000 0066 6600 4444"
  $"0000 FE44 FA00 35FA 0012 0333 3300 0003"
  $"3330 0011 1110 0011 1100 0888 88FD 0017"
  $"0777 7000 0777 7770 6666 6000 0666 6000"
  $"0666 6600 4444 4004 FE44 FA00 2FF9 00FE"
  $"330E 0033 3330 0011 1110 0111 1100 0888"
  $"80FD 0003 0777 7707 FE77 0C00 6666 6000"
  $"6666 6000 0666 6600 FA44 FA00 2CF9 00FB"
  $"330B 3000 1111 0001 1111 0008 8880 FD00"
  $"0007 FC77 0D70 0066 6600 0066 6660 0006"
  $"6660 00FA 4400 40FB 002D F900 0003 FC33"
  $"0B00 0011 1100 0111 1000 0888 80FC 00FC"
  $"770E 0000 6666 0000 6666 0000 0666 6000"
  $"04FB 4400 40FB 0033 F800 0003 FE33 0C30"
  $"0000 1111 0001 1110 0008 8880 FC00 0007"
  $"FE77 FE00 0C66 6600 0066 6600 0006 6660"
  $"0000 FE44 0300 0444 40FB 0002 C100 02C1"
  $"0002 C100 02C1 0002 C100 0BFE 0001 0FF0"
  $"E400 00FF E400 20FE 0001 0FF0 F700 010F"
  $"FFF1 0004 0FFF 000F F0F4 0000 FFFC 0004"
  $"0FF0 0000 FFFF 001B FE00 00FF F700 010F"
  $"FFF0 0004 0FFF 00FF F0EE 0004 0FF0 0000"
  $"FFFF 003F FE00 07FF FFF0 00FF 000F F0FE"
  $"0000 FFFE 000A 0FFF FFF0 0FFF F000 00FF"
  $"F0FD 0009 FFFF 0FFF F000 0FFF FF00 FEFF"
  $"090F FFFF F00F F000 0FFF 00FE FF04 0FFF"
  $"FFF0 003E FE00 0BFF F0FF 00FF 000F F000"
  $"000F F0FE 000A 0FFF 0000 FF00 FF00 0FF0"
  $"FFFD 00FE FF19 0FF0 00FF 00FF 00FF F000"
  $"0FFF 0000 0FF0 00FF 0FF0 00FF 0000 0FF0"
  $"FF00 40FE 0006 FF00 0FF0 FF00 FFFE 000F"
  $"0FF0 0FFF 000F F000 0FF0 00FF 00FF 00FF"
  $"FE00 1D0F F0FF FF0F F00F F000 FF00 FF00"
  $"000F F000 000F F00F F00F F000 FF00 000F"
  $"F0FF 0040 FE00 06FF 000F F00F FFF0 FE00"
  $"0F0F F0FF 0FF0 0FF0 000F FFFF F000 FF00"
  $"FFFE 001D 0FF0 FFF0 FF00 FF00 0FF0 00FF"
  $"0000 0FF0 0000 0FF0 FF00 0FF0 00FF 0000"
  $"0FF0 FF00 4009 0000 0FF0 000F F00F FFF0"
  $"FE00 0F0F F000 0FF0 FF00 000F F000 000F"
  $"F00F FFFE 000E 0FF0 FF00 FF00 FF00 0FF0"
  $"0FF0 0000 FFFE 000A FF00 FF00 0FF0 0FF0"
  $"0000 FFFE 003E 0800 000F F000 FF00 00FF"
  $"FC00 0EFF 00FF 0000 0FF0 0FF0 0FF0"
  $"FFFF FE00 0EFF 0000 0FF0 00FF 00FF F00F"
  $"F000 00FF FE00 0AFF 00FF 00FF 000F F000"
  $"00FF FE00 3E08 0000 0FFF FFF0 0000 FFFC"
  $"0004 0FFF F000 FFFE 0006 FFFF 0000 FFFF"
  $"F0FE 000E FF00 000F F000 0FFF FFF0 0FF0"
  $"0000 FFFE 000A FF00 0FFF F000 0FF0 0000"
  $"FFFE 000C FA00 010F F0EF 0001 0FF0 DE00"
  $"0DFA 0001 0FF0 F100 020F F0FF DD00 0BFA"
  $"0000 FFEF 0001 FFF0 DD00 02C1 0002 C100"
  $"02C1 0002 C100 02C1 0002 C100 02C1 0002"
  $"C100 02C1 0002 C100 02C1 0002 C100 02C1"
  $"0002 C100 0703 000F FFF0 C500 3215 00F0"
  $"000F 0000 F000 0FFF 000F FF00 0FFF 0000"
  $"0FFF FF0F FB00 03F0 0000 F0FC 0002 F000"
  $"F0FC 0008 0FFF F000 F000 00FF F0F7 0031"
  $"150F 00FF 00F0 0FF0 00F0 00F0 F000 F0F0"
  $"00F0 0000 0F00 0FFB 0003 FF00 0FF0 FC00"
  $"02F0 00F0 FB00 07F0 FF0F F000 0F00 0FF7"
  $"003F 1B0F 0F00 00F0 00F0 00F0 00F0 F000"
  $"F0F0 00F0 0000 0F00 0FFF 000F F000 00FD"
  $"F00D 0FF0 00FF 00F0 00F0 0FF0 0F00 0F00"
  $"FDF0 0D00 0F00 000F 0FF0 0FF0 0F00 F0FF"
  $"F041 0F0F 0F00 00F0 00F0 000F FFF0 0FFF"
  $"000F FFFE 002C 0F00 0F00 F0F0 0F00 00F0"
  $"0F00 F000 0F0F 00F0 FFFF F000 0F00 F0F0"
```


© The Definitive MacTutor, Vol. 4

```
  $"00F0 F000 F000 0F00 FF0F F000 F00F 0F00"
  $"F0F0 0F3D 060F 00FF 00F0 00F0 FE00 21F0"
  $"F000 F0F0 00F0 0000 0F00 0F00 F0FF FF00"
  $"00F0 0000 F00F FF0F 0000 F000 F00F FF00"
  $"0FFA 000C 0F00 0F0F 0000 F00F 0F00 F0F0"
  $"0F3E 1800 F000 0F00 00F0 0000 0F00 F000"
  $"F0F0 00F0 0000 0F00 0F00 F0F0 FE00 10F0"
  $"0000 F0F0 0F0F 00F0 F000 F0F0 0F00 F0F0"
  $"FB00 0C0F 000F 0F00 00F0 0F0F 00F0 F00F"
  $"3D0F 000F FFF0 0000 F000 0FF0 000F FF00"
  $"0FFF FE00 190F 000F 00F0 0FF0 0000 F000"
  $"00F0 0FFF 00FF 00F0 00F0 0FFF 0F00 0FFA"
  $"000B FFF0 0F00 000F F000 FFF0 FFF0 05C3"
  $"0001 F000 05C3 0001 F000 02C1 0002 C100"
  $"02C1 001F F600 10F0 00FF 0FF0 000F FFF0"
  $"00F0 0000 0F00 000F FC00 01FF FFEF 0000"
  $"F0F7 001F F600 08F0 000F 00F0 000F 000F"
  $"FD00 030F 0000 0FFC 0002 F000 F0F0 0000"
  $"F0F7 0031 F700 2B0F 0F00 0F00 F000 0F00"
  $"0F0F F00F FF0F FF00 FFF0 0FFF 0000 F000"
  $"F00F F000 FFF0 0FF0 0F0F F0F0 00F0 0FF0"
  $"00FF F0F7 0031 F700 130F 0F00 0F00 F000"
  $"0FFF F000 F0F0 0F0F 00F0 0F00 F0FE 0014"
  $"FFFF 00F0 0F0F 0000 F00F 0FF0 000F 0F00"
  $"F00F 0F00 F0F7 0031 F700 2BFF FFF0 0F00"
  $"F000 0F0F 0000 F0F0 0F0F 00F0 0F00 0FF0"
  $"0000 F0F0 00FF FF00 FF00 FFFF 0F00 000F"
  $"0F00 FFFF 0F00 F0F7 0031 F700 1AF0 00F0"
  $"0F00 F000 0F00 F000 F0F0 0F0F 00F0 0F00"
  $"000F 0000 F00F 00F0 FE00 03F0 F000 0FFE"
  $"0006 F000 F000 0F00 F0F7 0032 F700 21F0"
  $"00F0 0F00 F000 0F00 0F00 F00F FF0F 00F0"
  $"00F0 FFF0 0000 F000 F00F F00F FF00 0FF0"
  $"0FFE 0007 F000 0FF0 00FF F00F F800 06EA"
  $"0000 0FD9 0007 EB00 010F F0D9 0002 C100"
  $"02C1 0002 C100 02C1 0002 C100 00FF"
};

resource 'PICT' ( ( 0xC000 | ( DriverID << 5 ) ) +
                  bwChromaPicture ) {
  1023,
  {83, 167, 166, 295},
  $"1101 A000 8201 000A 0000 0000 02D0 0240"
  $"9800 1200 5300 A000 A601 2800 5300 A700"
  $"A601 2700 5300 A700 A601 2700 0002 EF00"
  $"02EF 0002 EF00 06FC 0000 3CF5 0008 FE00"
  $"020F E07C F500 08FE 0002 3FF0 7CF5 0008"
  $"FE00 027F F8F8 F500 08FE 0002 FEF8 F8F5"
  $"0011 0900 0001 F878 F001 E000 3FFD 0000"
  $"7FFE 0013 0F00 0001 F0F8 FFC1 FFC0 FF87"
  $"FE00 01FF E0FF 0013 0F00 0003 E0F9 FFE1"
  $"FFE1 FFC7 FF7E 03FF E0FF 0013 0F00 0003"
  $"E0F1 FFF0 FFE3 FFC7 FFFF 07F3 E0FF 0013"
  $"0F00 0003 C001 FFF0 FDE7 E3C7 EFFF 8FC3"
  $"E0FF 0013 0F00 0003 C001 FCF0 F807 C3C7"
  $"C7FF 8F83 C0FF 0013 0F00 0003 C001 FCF1"
  $"F00F 83C7 C7E7 9F07 C0FF 0013 0F00 0003"
  $"C001 F8F1 F00F 83CF 87C7 9F07 C0FF 0013"
  $"0F00 0003 C001 F8F1 E00F 03CF 8F87 9E0F"
  $"80FF 0013 0F00 0003 C001 F9F1 E00F 07CF"
  $"0F87 9E0F 80FF 0013 0F00 0003 E001 F1F3"
  $"E00F 07CF 0F07 9E1F 80FF 0013 0F00 0003"
  $"E0F1 F1E3 E00F 0F9F 0F0F 9F3F 80FF 0013"
  $"0F00 0001 F9F1 F3E3 C00F BF9F 1F0F 9FFF"
  $"80FF 0013 0F00 0001 FFF1 E3E3 C00F FF1E"
  $"1F0F 0FFF C0FF 0012 FE00 0CFF E1E3 C3C0"
  $"07FE 1E1E 0F0F FFC0 FF00 12FE 000C 3FC1"
  $"E3C3 C003 F81E 1E0F 07E3 C0FF 0002 EF00"
  $"02EF 0002 EF00 02EF 0002 EF00 0901 0003"
  $"FA00 0018 F900 1305 0003 0000 0380 FE00"
  $"0838 C000 0006 0030 6000 1104 0006 0000"
  $"0EFD 0001 39C0 FD00 0230 6000 1311 0007"
  $"C630 180F CF07 007B C3E7 EFCC 39FB F000"
  $"1311 0007 6630 300E 198D 807E C667 0E0C"
  $"6C60 C000 1311 0006 3660 338C 3199 80DE"
  $"CC66 0C0C CC60 C000 1311 0006 33C0 36CC"
  $"3F19 80DD 98C6 0C0D 8C60 C000 1311 000C"
  $"33C0 30D8 3033 80D9 98CC 1819 8CC1 8000"
  $"1311 000C 6180 1998 3337 8183 19CC 1819"
  $"98C1 8000 1311 000F C180 0F18 1E1F 0183"
  $"0FCC 1818 F0C1 8000 0A02 0000 03FD 0000"
  $"03F7 000A 0200 0003 FD00 0036 F700 0A02"
  $"0000 06FD 0000 1CF7 0002 EF00 02EF 0002"
  $"EF00 02EF 0002 EF00 02EF 0002 EF00 02EF"
  $"0002 EF00 02EF 0002 EF00 02EF 0002 EF00"
  $"02EF 0005 0100 3CF1 0012 0E00 4210 E38E"
  $"0FA0 0104 0044 003C 41C0 FE00 120E 0099"
  $"3114 5102 2001 8C00 4400 16C2 20FE 0013"
  $"1100 A111 1451 0238 C154 C644 C895 420B"
  $"325C 0013 1100 A110 F38E 0225 2124 297C"
  $"2514 426C 4A52 0013 1100 9910 1451 0225"
  $"E104 E844 E200 0228 4A52 0013 1100 4210"
  $"2451 0225 0105 2945 2500 0228 4A52 0013"
  $"1100 3C10 C38E 0224 C104 E644 E880 01C8"
  $"31DC 0005 F100 0110 0005 F100 0110 0002"
  $"EF00 02EF 0002 EF00 10FE 0006 046C 3C40"
  $"8200 78FD 0000 40FE 0010 FE00 0604 2422"
  $"0082 0044 FD00 0040 FE00 11FE 000B 0A24"
  $"22CE E738 44C7 32D1 31C0 FE00 11FE 000B"
  $"0A24 3C52 9240 7928 4B0A 4A40 FE00 11FE"
  $"000B 1F24 2852 9230 51E6 7A0A 7A40 FE00"
  $"11FE 000B 1124 2452 9208 4901 4204 4240"
  $"FE00 11FE 000B 1124 224E 9170 44CE 3204"
  $"31C8 FE00 06FB 0000 02F6 0006 FB00 000C"
  $"F600 02EF 0002 EF00 02EF 0002 EF00 02EF"
  $"00A0 0083 FF"
};

resource 'PICT' ( ( 0xC000 | ( DriverID << 5 ) ) +
                  sorryPicture ) {
  521,
  {328, 367, 366, 461},
  $"1101 A000 8201 000A 0000 0000 02D0 0240"
  $"9800 0E01 4801 6801 6E01 D001 4801 6F01"
  $"6E01 CD01 4801 6F01 6E01 CD00 000F 0600"
  $"3C07 0000 C070 FE00 030C 00C0 000F 0600"
  $"1803 0000 CC30 FE00 030C 00C0 000D 0600"
  $"1803 0000 CC30 FC00 01C0 000F 0B00 181F"
  $"3DF8 1E3E 783E F3DD F8FF 000F 0B00 1833"
  $"66EC 0C3B CC67 9E6C ECFF 000F 0B00 1833"
  $"66CC 0C33 FC79 FFEC CCFF 000F 0B00 1833"
  $"66CC 0C33 C01F 860C CCFF 000F 0B00 1837"
  $"66CC 0C33 CC67 9E6C CCFF 000F 0B00 3C1F"
  $"BDFE 073E 787C F3DF FEFF 0002 F300 02F3"
  $"0002 F300 02F3 000B FC00 001C FE00 01E1"
  $"C0FE 000B FC00 050C 0001 8060 C0FE 000B"
  $"FC00 050C 0001 8060 C0FE 000F 0B00 03F1"
  $"E0F3 CCF7 E3DE 7CCF 3EFF 000F 0B00 01DB"
  $"3196 6D9B E1B3 76D9 E6FF 000F 0B00 019B"
  $"3186 6D9B 018F 66DF F8FF 000F 0B00 019B"
  $"3186 6D9B 01BB 66D8 1EFF 000F 0B00 019B"
  $"3196 6D9B 01B7 66D9 E6FF 000F 0B00 03FD"
  $"E0F3 DEF7 80FF FDEF 7CFF 0002 F300 02F3"
  $"0002 F300 02F3 000C 0800 0001 CC00 0E00"
  $"00E0 FC00 0C08 00C0 00CC 0006 0000 60FC"
  $"000C 0800 C000 C000 0600 0060 FC00 0F0B"
  $"01EF 07DC FFE6 7BFC 7C7B F780 FF00 0F0B"
  $"00D9 8CCD 9BB6 CD98 76CD FCC0 FF00 0F0B"
  $"00D9 8CCD E336 3D98 66FD 8FC0 FF00 0E0A"
  $"00D9 8CCC 7B36 ECF0 66C1 8CFE 000F 0D00"
  $"D98D CD9B 36DC F066 CD8C D998 000F 0D00"
  $"7F07 FFF3 EF7E 60FF 7BC7 9998 0009 FD00"
  $"0303 0000 60FB 0009 FD00 0303 0003 C0FB"
  $"0009 FD00 0307 8003 80FB 00A0 0083 FF"
};
```

Macintosh II
The Palette Manager & Color Puzzle

Steven C. Sheets
Contributing Editor
MacTutor Vol. 4 No. 4

For the next two months, this column is going to be a little different than the previous ones. Normally a new area of Mac / / programming is picked each month. That area is discussed, along with it's implementation and implications, and a sample program is given that demonstrates the concepts in action.

This month's (much promised) column is on the Palette Manager, while next month's will be on using the Palette Manager to do Palette Animation. In this article, there will be a explanation of why the Palette Manager is needed and how it works. The basic Palette Manager calls will be explained. Next month, the sample program, PalFun, will be taken apart step by step, explaining the Palette Manager Animation calls and the special effect done using Palette Animation.

Still, every column should have at least one example of code, so presented here is Color Puzzle. It was created back in the days when there was very little truly useful color applications or Desk Accessories. Two Mac-Fanatics wanted to show everyone the real reason they purchase their Mac //s. Their idea of programming was telling this Author that the Color Puzzle was a needed tool. This Author agrees with them!

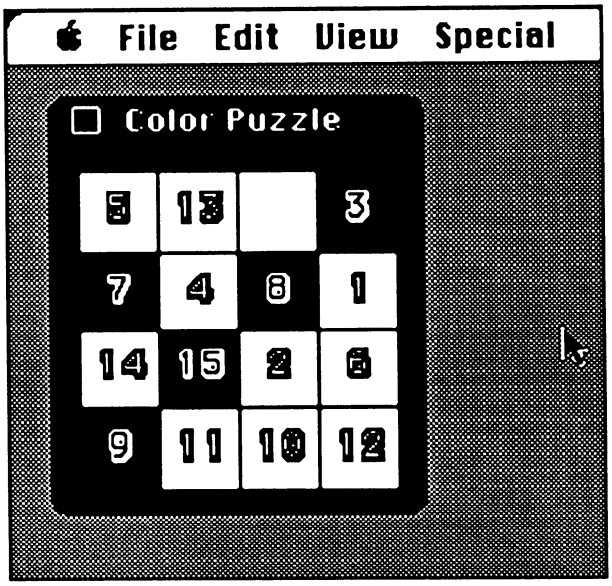


Fig. 1 Our color puzzle in black and white! Pressing the option key brings up the color chooser.

RGB vs Pixel Space

So far almost all of the drawing explained in this column has been in RGB space. A program picks a specific color, consisting of an 8 bit Red component, an 8 bit Green component and an 8 bit Blue component. The program then draws with this color, using Color Quickdraw commands, to a specific Color Graphics Device (ex. Apple's Mac // Video Card/Monitor). While most Graphics Devices can display any possible RGB color, most Devices can only display a finite number of colors at any time. This is due to the way the memory of the Mac // Video Devices (ie. Screen) is structured.

Every pixel on the screen has an associated spot in memory. On the older non-Color Macs, this spot in memory was a single bit that designated if the pixel was white or black. On the color Mac //, this spot in memory is a variable size, unsigned integer. The integer contains an index number. That number corresponds to a table of RGB colors (called a Color Look Up Table or CLUT) that the Video card maintains and uses to physically display the screen. If a pixel has a index of 3 in memory, that same pixel is displayed on the Video Card/Monitor using the 3rd color in the Video Card's CLUT. Changing the pixel index, changes the pixel color on the screen. Color Quickdraw does it's drawing by manipulation of each pixel's index.

Obviously the number of colors most card can display is dependent on the amount of memory used for each pixels; 2 to the power of the number of bits per pixels is equal to the number of colors that can be displayed at one time. For example, the initial Apple Video Card has 4 bits per pixels and can only display 16 colors at one time. If someone wants to upgrade the memory on the card using Apple's Video Expansion Kit to 8 bits per pixel, then the card can display up to 256 colors at one time. 8 Bits per pixel (256 Colors) is currently the most common configuration of Mac // Graphic Cards, though 24 Bit Cards may be just around the corner.

The decision of what colors are to be physically displayed on a card and what RGB values map into which physical color is handled by the Color Manager (not the Quickdraw Manager). When the Mac // is initially booted, the Color Manager normally sets the colors of each Graphics Device to commonly used colors. The Mac //'s default color environment (ie. settings of the CLUT) are sometimes called the "Apple System Colors". When Color Quickdraw needs to know what index corresponds to the RGB value it wants to draw with, it calls the Color Manager. The Color Manager checks the current status of the Video Card's CLUT and returns an index value. This is the index of the RGB value closest to the RGB color Color Quickdraw wants to draw with. There may not be a RGB color in the Video Card's CLUT anywhere close to the requested color. The Color Manager returns the best

© The Definitive MacTutor, Vol. 4

choise given the current status of the CLUT. For many applications, best choice is good enough. However it is not hard to think of applications that need exact control of the colors used. If the program is an elaborate Drawing or Painting program, the user may want to control the colors himself, allowing more greens for a landscrape, or more flesh tones for a portrait. If the program is a game, the various figures and images may have to be in a specific color. If the program is connected to a video input scanner, the program may want to set the colors of the card to match the colors used by the scanner. The Color Manager allows direct manipulation of each pixel's colors.

Color Manager vs Palette Manager

After praising the Color Manager for what it can do, the next statement may sound a little strange; NEVER use the Color Manager! Instead use the Palette Manager. The Color Manager would be adequate to use if all programs were written to be the only thing running on a specific system configuration. This is almost never the case. Most programs have to be written to run on any number or size of video cards. Other code segments may be running at the same time that require different colors. Obviously programs under Multifinder needs to share the colors. Even Desk Accessories may need specific colors. Who decides which colors are needed? More importantly, when there is a limited number of bits on the video card, who decided which colors are safe not to include (more than one code segment may want the same color). If video card's colors change, how does the rest of the system handle this? The Palette Manager is the solution for these problems.

When using the Color Manager, the Manager is manipulating the entire Color Table on a Specific Card (cumbersome). When using the Palette Manager, the Manager is concerned with groups of colors (ie. a Palette) that each specific window needs to display. The Palette Manager calls the Color Manager to select the best choice of colors for each video card (no matter it's size) that would make all the Palettes happy.

To paraphrase a quote, "You can not make all the palettes happy all the time". However, a program <u>can</u> make the user happy all the time. Each Palette is associated with a given window. It is logical to give the top most Window (and it's associated Palette) the highest priority. This is the window the user is currently looking at. If the top most Palette is smaller than the number of colors on a card, other Palettes are used to select the remaining colors. If there are not enough colors for all the Palettes, the top most window has it's colors selected first.

Using the Palette Manager

There are many ways to create a Palette. The Color Puzzle uses the easiest method to understand. First a Palette Handle is created using **NewPalette**. NewPalette needs to know the number of entries the Palette will have, the color table it should use to pre-set the colors, and the Usage and Tolerance of the colors. Note: Entries are not really colors at this point as much as there are RGB values that are requested. They are number from 0 (first entry) to N-1 (where there are N number of entries in the Palette).

Color Tables are lists of RGB values that the Mac // uses to set various graphic data structures. When passed to NewPalette, the Palette entries are pre-set using the RGB colors in the table. Once set, the Color Table is not used by the Palette Manager or by that Palette Handle, and can be safely disposed of. If NIL is passed for the color table, the entries in the Palette are set to black. There are no Quickdraw calls or methods to create a color table other than stuffing the values into a handle. Thus NewPalette is normally passed NIL, and the colors are changed after the Palette is created, but before it is attached to a window.

The Colors of the Palette can be set using the **SetEntryColor** call. SetEntryColor changes the RGB values of a specific entry in a Palette. SetEntryColor is passed the Palette Handle, an index to the entry (0 being the first entry, 1 being the second and so on), and the RGB value to change the entry to. **GetEntryColor** is the reverse call to retrieve a specific RGB value from an entry. It is passed the Palette Handle, an index to the entry being referenced, and the RGB variable to return the values in.

No matter which method is used to set the Colors of the Entry, always set entry 0 to White (RGB $FFFF,$FFFF,$FFFF) and entry 1 to Black (RGB $0000,$0000,$0000). Do not set any other entries to exactly Black or White. If a program needs an almost black or white shade, use near misses (ex. $0001,$0000,$0000 or $FFFE,$FFFF,$FFFF). To the eye, they will be black or white, but will still be legal to the Palette Manager.

Besides the actual RGB value, each entry has a Usage and Tolerance value associated with it. These values effect exactly how the Palette Manager selects which colors to display in a given instance. There are 4 type of Color Usage; Tolerant, Courteous, Explicit and Animation. Each Usage has a predefined constant associated. Each entry's Usage value must be set to one of these constants. The use of an entry's Tolerance value is depenent on the Usage setting. Notice that difference between the terms "Tolerance" and "Tolerant". The Tolerance value is an attribute of an entry, while the Tolerant constant is one of the four type of Usage colors.

A Tolerant Color is used when the window needs to draw a relatively exact RGB Color. How close of a match is needed between the asked for RGB color and the actual color drawn is dependent on the value of Tolerance. A Tolerance of 0 mean the program wants an exact match. If that exact color is not avaliable on the Video Card, then the Palette Manager will change the colors of the card to get that color. If Tolerance is a number, the Palette Manager will check the currently displayed colors on the card. If there is one that is close to the entries RGB value (ie. one where any given difference between the Red, Green or Blue components are less than or more than Tolerance), the Palette Manager will use that color and not reset the card. The majority of the entries in most Palettes have a Usage value of 2 (Tolerant).

A Courteous Color is a place holder. The Tolerance value is ignored. The Palette Manager will not change the colors of a

graphic card to get this color. The Manager will deallocate (ie. change that color to a more needed one) other colors first when it is trying to find unused colors. In other words, when talking strictly about Courteous Colors and Color with no usage, the Palette Manager will deallocate RGB colors from a Video Card's CLUT that are non-usage Colors, before Courteous Colors. Tolerent color will always be deallocated last. Using Courteous Color is also an useful way to store RGB values that the program may want to use (see below).

An Explicit Color is a convenience for programs that want to show what the current color environment is. Instead of having to use the Color Manager to find out the current color environment, a program creates a Palette of colors of usage Explicit (Colors and Tolerance ignored). When drawing with Entry number 0, instead of drawing the RGB value of stored at Entry 0, the color 0 of the Color Device is displayed. This is an ecellent method to show the current CLUT colors. Next Months PalFun has a window that does this. There are a number of Public Domain Desk Accessories that do exactly this. Such tools are invaluable to any programmer who is working with the Palette Manager.

An Animated Color is special tool used for Palette Animation (more next month). For now, realize that when an Animated Color is allocated, that color will only appear on the Window that has the Palette that the color is allocated to. If one Palette allocated a Red Animated Entry (RGB $FFFF,$0000,$F0000), and then another Palette allocated a Red Tolerent Entry (Tolerance 0), another Red would be added to the Video Cards color list instead of using the first colors. When a RGB color is drawn in a window, if that exact color is in the CLUT of the Video Card as an Animated Color, another index will be selected.

Usage and Tolerance are set by NewPalette for the entire Palette or each entries Usage and Tolerance can be set with the **SetEntryUsage**. SetEntryUsage is passed the Palette Handle, the entry to change, and the Usage and Tolerance values to change the entry to. **GetEntryUsage** (not used in PalFun) is the reverse call to retrieve a specific Usage and Tolerance values from an entry. It is passed the Palette Handle, the entry to look at, and the Usage and Tolerence variables to return the values in.

One more thing should be mentioned when talking about selecting colors for a Palette. All things being equal, when the Palette Manager wants to set the video colors of a color card (from the current Palette or another Palette, if there are enough colors on the video card), it selects them in a first-come-first-serve order. If there are only enough unalocated colors on the Color Card for half the palette, the first half gets set and the second must must match to the closest colors. If the colors in the Palette were not distributed evenly (ie. All the Oranges first, then then Purples then the Yellow), there would be many shades of the First colors (ie. Orange) and not many shades of the remaining. It is best to distribute the colors evenly over the Palette. Place the best 16 colors first (don't forget color 0 should be white, and color 1 should be black) so that the window will appears it's best on a 4 Bit card. If the Palette is the second window, it may get an odd number of color slots given to it. Make the ones given work! Select all the entries so that no matter how many entries are selected, the window will appear at it's best.

Once the Palette is created using NewPalette, with or without SetEntryColor, **SetPalette** is used to connect a specific window to the Palette. SetPalette makes a logical connection between the Palette and the Window (ie. This Palette goes with this Window). The call is passed the window pointer, the palette handle, and a boolean flag to indicate if this window needs to be updated whenever the colors change.

To understand the need for an update, imagine what happens when another Window (and it's Palette) is selected. If there were enough colors on the video card for all windows to get the Colors they need, nothing happens, since the best colors are already selected. If this is not the case (ie. small number available colors on the card and a large number of colors in the window), the Palette Manager changes the video card's colors to reflect the change in Palette priority. On the video card, RGB values that are used to display some pixels a certain color are suddenly changed. A globe that was a shade of green before, may suddenly be a shade of brown. There still may a different green that would be a better green to draw with, so the window should be redrawn. This way, the window will look it's best no matter what the current color environment is. Generally all Windows with associated Palettes need to be redrawn when the color selection changes. If the redraw takes too long, or the window has only Animated Colors (more next week on why), then the update flag does not need to be set. Remember this update event is caused by the Palette Manager; a program does not have to have any special code to enact it.

If at any time in the program after the Palette is associated with a window (using SetPalette), an Palette entry's Color, Usage or Tolerance is changed using SetEntryColor or SetEntryUsage, **ActivatePalette** (not used in PalFun) needs to be called. This routine is normally called by the Window Manager whenever window status changes (ie. new window comes to front). ActivatePalette is simply passed the window of the Palette that changed, and it will check and handle changing any color on any color cards.

When drawing to a Window that has a Palette associated with it, normal Color Quickdraw commands can be used. This involves setting numerous RGB variables and passing them to Quickdraw calls (ex. Red Globe, Green Globe). Then Color Quickdraw calls the Color Manager to pick a close index. Usually the window already knows which colors to use. The entries in the Palette for each window are the colors that are usually drawn on the windows. **PmForeColor** and **PmBackColor** are just like their Color Quickdraw counterparts (RGBForeColor and RGBBackColor); they set the Foreground color and Background color of a Color Port. They are passed the number of some entry in the Palette and set the foreground or background color to the RGB value (ex. Blue Window). This is the only way that an Animated Color or an Explicit Color can be drawn with. Even if the Palette Manager is not used in a program to change the environment, the use of Courteous Colors and the PmForeColor/PmBackColor calls can simplify coding.

Calling NewPalette, SetEntryColor and SetPalette is a good method to use if the RGB entries have to be calculated at run time, and are not known in advance. However there is a simpler method if the colors for a window/Palette are known in advance. Create the Window using a 'WIND' resource and the GetWindow call. Also create a resource where the type is 'pltt' and the ID number is the same as the 'WIND' resource. This 'pltt' resource has the Palette information for the Window. When the Window is created in a program, the associate Palette is automatically created and attached to the window. Using this Method with the PmForeColor/PmBackColor calls place the colors of some applications outside the code section for easy editing with ResEdit or some other tool (similar to the reason most programmers place the Text information outside the code section). The format of a 'pltt' resource is:

Data	Size
Number Colors	2 Bytes (INTEGER)
Internal Data	14 Bytes (set 0)
(for each Color)	
Color	6 Bytes (RGB Value/3 INTEGERS)
Usage	2 Bytes (INTEGER)
Tolerance	2 Bytes (INTEGER)
Internal Data	6 Bytes (set 0)

Color Puzzle

The sample code, Color Puzzle, use the Palette Manager to help maintain the colors of each square. It functions like the traditional Puzzle from Apple with some added features. Opening the Desk Accessory with the Option Key down creates a BIG puzzle (for those long phone calls). Clicking on a Square with the Option Key Down will allow the user to change the colors used by the Puzzle. These changes are also saved to disk, so the user will have to live with what he created. Enjoy!

Last Comments

Thanks for all the comments about the various areas/programs discussed in this column. This Author is always open to contacts from other Mac programmers!

PalFun, next month's sample program, uses most of the Palette Manager calls explained here. It also uses many of the Animation calls that will be covered next month. In the meantime, experiment! To develop a feel for how the Palette Manger works, it is best to manipulate various windows and see how the Palette Manager reacts.

```
{     Color Puzzle DA by Steve Sheets 12/09/87    }
{  Desk Accessory to show Puzzle in Color on Mac //.
}
UNIT ColorPuzzleDA;

INTERFACE

   FUNCTION Main (theDCE : DCtlPtr;
                  IOPB : ParmBlkPtr;
                  driveCall : Integer) : OSErr;

IMPLEMENTATION

   CONST
      DriverOpen = 0;
      DriverPrime = 1;
      DriverControl = 2;
      DriverStatus = 3;
      DriverClose = 4;

      dCtlEnable = 10;

      kLine = 18;

      Edge = 10;
      xSize = 30;
      ySize = 30;
      vOff = -12;

      kxTiles = 4;
      kyTiles = 4;

      optxTiles = 10;
      optyTiles = 10;

      MaxTiles = 10;
      MaxNum = 100;
      RGBMax = 102;

      ResT = 'RGBs';

      pmTolerant = $0002;

      HelpWait = 240;
      Flash = 60;

      kWhite = 0;
      kBlack = 1;

      ScrCount = 2000;
      OptScrCount = 10000;

(DA internal variable stored as a single Handle (the Picture,
the Window)
(  and a Rectange the size of the Picture).)
   TYPE
      BoardData = PACKED ARRAY[1..MaxNum] OF integer;

      PuzzleData = RECORD
            myPal : handle;
      myID, xBig, yBig, numTiles, xTiles, yTiles : integer;
            PuzzleWindow : WindowPtr;
            ColorFlag : Boolean;
            Normal : boolean;
            Board : BoardData;
         END;
      PuzzlePtr = ^PuzzleData;
      PuzzleHandle = ^PuzzlePtr;

      RGBColor = RECORD
            red : INTEGER; { magnitude of red }
            green : INTEGER; { magnitude of green }
            blue : INTEGER; { magnitude of blue }
         END;

(Format of RGB resource where colors are stored between
boots.)
      RGBData = PACKED ARRAY[2..RGBMax] OF RGBColor;
      RGBPtr = ^RGBData;
      RGBHandle = ^RGBPtr;

(Inline Palette Manager & Color Picker calls)

   FUNCTION GetColor (where : Point;
                      prompt : Str255;
                      inColor : RGBColor;
```

```
                    VAR outColor : RGBColor) : BOOLEAN;
  INLINE
     $3F3C, $0009, $A82E;

  PROCEDURE PmForeColor (dstEntry : INTEGER);
  INLINE
     $AA97;

  PROCEDURE ActivatePalette (srcWindow : WindowPtr);
  INLINE
     $AA94;

  PROCEDURE PmBackColor (dstEntry : INTEGER);
  INLINE
     $AA98;

  FUNCTION NewPalette (entries : INTEGER;
              srcColors : handle;
              srcUsage, srcTolerance : INTEGER) : handle;
  INLINE
     $AA91;

  PROCEDURE SetPalette (dstWindow : WindowPtr;
                        srcPalette : handle;
                        cUpdates : BOOLEAN);
  INLINE
     $AA95;

  PROCEDURE DisposePalette (srcPalette : handle);
  INLINE
     $AA93;

{Create a Color Window.}
  FUNCTION NewCWindow (wStorage : Ptr;
                       boundsRect : Rect;
                       title : Str255;
                       visible : BOOLEAN;
                       procID : INTEGER;
                       behind : WindowPtr;
                       goAwayFlag : BOOLEAN;
                       refCon : LONGINT) : WindowPtr;
  INLINE
     $AA45;

  PROCEDURE SetEntryColor (dstPalette : Handle;
                           dstEntry : INTEGER;
                           srcRGB : RGBColor);
  INLINE
     $AA9C;

  PROCEDURE GetEntryColor (srcPalette : Handle;
                           srcEntry : INTEGER;
                           VAR dstRGB : RGBColor);
  INLINE
     $AA9B;

{Procedure to check if move is legal,  if so, returns move.}
  PROCEDURE DoDir (Dir, xTiles, numTiles : integer;
                   VAR UnLegal : boolean;
                   VAR CT : integer);
     VAR
        rw : integer;
  BEGIN
     UnLegal := false;
     rw := ((CT - 1) DIV xTiles);
     CASE Dir OF
        1 :
           BEGIN
              CT := CT - 1;
     IF (((CT - 1) DIV xTiles) <> Rw) OR (CT < 1) THEN
              UnLegal := true;
           END;
        2 :
           BEGIN
              CT := CT + 1;
              IF (((CT - 1) DIV xTiles) <> Rw) OR (CT >
numTiles) THEN
                 UnLegal := true;
           END;
        3 :
           BEGIN
              CT := CT - xTiles;
              IF (CT < 1) THEN
                 UnLegal := true;
           END;
        4 :
           BEGIN
              CT := CT + xTiles;
              IF (CT > numTiles) THEN
                 UnLegal := true;
           END;
        OTHERWISE
           UnLegal := true;
     END;
  END;

{DA Open routine:  }
  FUNCTION Open (theDCE : DCtlPtr) : OSErr;
     VAR
        theE : OSErr;
        thePuzzle : PuzzleHandle;
        OldPort : GrafPtr;

{Returns true if the Mac had Color Quickdraw.}
     FUNCTION ColorQDExists : boolean;
        CONST
           ROM85Loc = $28E;
           TwoHighMask = $C000;
        TYPE
           WordPtr = ^INTEGER;
        VAR
           Wd : WordPtr;
     BEGIN
        Wd := POINTER(ROM85Loc);
        ColorQDExists := (BitAnd(Wd^, TwoHighMask) = 0);
     END;

{Is Option Key not Down?}
     FUNCTION IsOptNotDown : BOOLEAN;
        VAR
           theMap : KeyMap;
     BEGIN
        GetKeys(theMap);
        IsOptNotDown := NOT BitTst(@theMap[1], 29);
     END;

{Scrambles the Puzzle info.}
     PROCEDURE ScrambleBoard; {LATER}
        VAR
        count, Dir, CurPos, NewPos, HoldPos, Sc : integer;
        UnLegal : boolean;
     BEGIN
        WITH thePuzzle^^ DO
           BEGIN
              FOR count := 1 TO numTiles DO
                 Board[count] := count;

              CurPos := numTiles;
              IF Normal THEN
                 sc := ScrCount
              ELSE
                 sc := optScrCount;
              FOR count := 1 TO sc DO
                 BEGIN
                    Dir := (Random MOD 4) + 1;
                    REPEAT
```


© The Definitive MacTutor, Vol. 4

```
                        NewPos := CurPos;
                        DoDir(Dir, xTiles, numTiles,
UnLegal, NewPos);
                        IF UnLegal THEN
                           BEGIN
                              Dir := Dir + 1;
                              IF Dir > 4 THEN
                                 Dir := 1;
                           END;
                     UNTIL NOT UnLegal;

                     HoldPos := Board[CurPos];
                     Board[CurPos] := Board[NewPos];
                     Board[NewPos] := HoldPos;
                     CurPos := NewPos;
                  END;
            END;
      END;

{Handle Opening already open windows (ie. either move the DA
to the front)
(of the screen or scramble the board).}
      PROCEDURE DAReset;
         VAR
            tmpRect : Rect;
      BEGIN
         thePuzzle := PuzzleHandle(theDCE^.dCtlStorage);
         HLock(theDCE^.dCtlStorage);

         WITH thePuzzle^^ DO
            BEGIN
               IF PuzzleWindow = FrontWindow THEN
                  BEGIN
                     ScrambleBoard;
                     SetPort(PuzzleWindow);
                     SetRect(tmpRect, 0, 0, xBig, yBig);
                     InvalRect(tmpRect);
                  END
               ELSE
                  SelectWindow(PuzzleWindow);
            END;
         HUnLock(theDCE^.dCtlStorage);
      END;

{Initialize the DA's internal data (including creating the
Data Handle, the Window)
(  (Color if needed) and the Rectange the size of the Pic-
ture)}
      PROCEDURE DAInit;
         VAR
            tmpRect : Rect;
            insideWindow : WindowPeek;

{Set Puzzle Pal entry to this RGB color}
         PROCEDURE SetPuzzlePal (N, R, G, B : integer);
            VAR
               C : RGBColor;
         BEGIN
            C.red := r;
            C.green := g;
            C.blue := b;
            SetEntryColor(thePuzzle^^.MyPal, n, C);
         END;

{Sets the Colors for Puzzle.}
         PROCEDURE SetPuzzleColor;
            VAR
               count : integer;
               C : RGBColor;
               RGBH : RGBHandle;
         BEGIN
            WITH thePuzzle^^ DO
               BEGIN
                  SetPuzzlePal(kWhite, $FFFF, $FFFF, $FFFF);
                  SetPuzzlePal(kBlack, 0, 0, 0);

                  RGBH := POINTER(GetResource(ResT, MyID));
                  IF RGBH = NIL THEN
                     BEGIN
                        FOR count := 2 TO numTiles DO
                           CASE count MOD 6 OF
                              0 :
                           SetPuzzlePal(count, $FFFF, 0, 0);
                              1 :
                           SetPuzzlePal(count, 0, $FFFF, 0);
                              2 :
                           SetPuzzlePal(count, 0, 0, $FFFF);
                              3 :
                           SetPuzzlePal(count, $FFFF, $FFFF, 0);
                              4 :
                           SetPuzzlePal(count, 0, $FFFF, $FFFF);
                              5 :
                           SetPuzzlePal(count, $FFFF, 0, $FFFF);
                              OTHERWISE
                           END;
                     SetPuzzlePal(numTiles + 1, $8888, $8888, $8888);
                     SetPuzzlePal(numTiles + 2, $FFFF, 0, $FFFF);
                     END
                  ELSE
                     BEGIN
                        FOR count := 2 TO numTiles + 2 DO
                           BEGIN
                              C := RGBH^^[count];
                        SetPuzzlePal(count, C.Red, C.Green, C.Blue);
                           END;
                        ReleaseResource(POINTER(RGBH));
                     END;
               END;
         END;

BEGIN
theDCE^.dCtlStorage := NewHandle(SizeOf(PuzzleData));
HLock(theDCE^.dCtlStorage);
thePuzzle := PuzzleHandle(theDCE^.dCtlStorage);

   WITH theDCE^, thePuzzle^^ DO
      BEGIN
         myID := theDCE^.dCtlRefNum;
         IF myID > 0 THEN
            myID := -myID;
         myID := $C000 + 32 * (-1 - myID);

         Normal := IsOptNotDown;
         IF Normal THEN
            BEGIN
               xTiles := kxTiles;
               yTiles := kyTiles;
            END
         ELSE
            BEGIN
               myID := myID + 1;

               xTiles := optxTiles;
               yTiles := optyTiles;
            END;
         numTiles := xTiles * yTiles;
         xBig := (xTiles * xSize) + (2 * Edge);
         yBig := (yTiles * ySize) + (2 * Edge);

         ColorFlag := ColorQDExists;

         tmpRect.top := 40;
         tmpRect.left := 40;
         tmpRect.right := tmpRect.left + xBig;
         tmpRect.bottom := tmpRect.top + yBig;
```

```
                IF ColorFlag THEN
                   BEGIN
                   PuzzleWindow := NewCWindow(NIL, tmpRect,
'Color Puzzle', true, rDocProc, Pointer(-1), True, 0);
                   myPal := NewPalette(numTiles + 4, NIL,
pmTolerant, 0);
                   SetPuzzleColor;
                   SetPalette(PuzzleWindow, myPal, true);
                   END
                ELSE
                   BEGIN
                      PuzzleWindow := NewWindow(NIL, tmpRect,
'B/W Puzzle', true, rDocProc, Pointer(-1), True, 0);
                      myPal := NIL;
                   END;
                ScrambleBoard;
                insideWindow := WindowPeek(PuzzleWindow);
                insideWindow^.windowKind := dCtlRefNum;
                dCtlWindow := Ptr(PuzzleWindow);
             END;
          HUnLock(theDCE^.dCtlStorage);
       END;

{Main Body of the Open Routine.  If the DA is not open,
initialize Puzzle.}
{  If there is a puzzle, scamble it.}
   BEGIN
      GetPort(oldPort);
      theE := noErr;

      IF theDCE^.dCtlStorage = NIL THEN
         DAInit
      ELSE
         DAReset;

      Open := theE;
      SetPort(oldPort);
   END;

{DA's donothing Prime Routine.}
   FUNCTION Prime (theDCE : DCtlPtr;
                           IOPB : ParmBlkPtr) : OSErr;
   BEGIN
      Prime := noErr;
   END;

{DA's Control Routine: Handles Update & Mousedown events.  If
the event is an}
{  Update, draw the Puzzle.  If it is a mousedown in the
contents of the window,}
{  check if the option is down.  If so, reset that Tiles
color.  If not, see if that}
{  is a legal move (and move if it is).  If the event is a
keydown, and a help key}
{  (?,/,h,H,Help) was pressed, display the dedication (who
needs help for}
{  a Puzzle?).}
   FUNCTION Control (theDCE : DCtlPtr;
                           IOPB : ParmBlkPtr) : OSErr;
      CONST
         accEvent = 64;
         accUndo = 68;
         accCut = 70;
         accCopy = 71;
         accPaste = 72;
         accClear = 73;
      VAR
         theEvent : EventRecord;
         thePuzzle : PuzzleHandle;
         aWindow : WindowPtr;
         tmpPat : Pattern;
         oldPort : GrafPtr;

{Given count, return Rect holding count.}
      PROCEDURE GetCountRect (n : integer;
                              VAR R : rect);
         VAR
            xn, yn : integer;
      BEGIN
         WITH thePuzzle^^ DO
            BEGIN
               xn := (((n - 1) MOD xTiles) * xSize) + edge;
               yn := (((n - 1) DIV xTiles) * ySize) + edge;
            SetRect(R, xn + 1, yn + 1, xn + xSize, yn + ySize);
            END;
      END;

{Given count, ColorFlag & Rect, draw Tile there.}
      PROCEDURE DrawTile (N : integer;
                          VAR R : rect);
         VAR
            S : str255;
      BEGIN
         WITH thePuzzle^^ DO
            BEGIN
               IF (N = numTiles) AND (NOT ColorFlag) THEN
                  BEGIN
                     EraseRect(R);
                     StuffHex(@tmpPat, '8800220088002200');
                     FillRect(R, tmpPat);
                  END
               ELSE
                  BEGIN
                     IF ColorFlag THEN
                        PmBackColor(N + 1);

                     EraseRect(R);

                     IF (N <> numTiles) THEN
                        BEGIN
                           NumToString(N, S);
                           MoveTo(R.left + ((xSize -
StringWidth(S)) DIV 2), R.bottom + vOff);
                           DrawString(S);
                        END;
                  END;
            END;
      END;

{Draws the board.}
      PROCEDURE DrawBoard;
         VAR
            count, yn, xn : integer;
            tmpRect : rect;
      BEGIN
         WITH thePuzzle^^ DO
            BEGIN
               IF ColorFlag THEN
                  PmForeColor(numTiles + 2)
               ELSE
                  BEGIN
                     StuffHex(@tmpPat, 'DD77DD77DD77DD77');
                     PenPat(tmpPat);
                  END;
               SetRect(tmpRect, 0, 0, xBig, edge);
               PaintRect(tmpRect);
               SetRect(tmpRect, 0, 0, edge, yBig);
               PaintRect(tmpRect);
            SetRect(tmpRect, xBig - edge + 1, 0, xBig, yBig);
               PaintRect(tmpRect);
            SetRect(tmpRect, 0, yBig - edge + 1, xBig, yBig);
               PaintRect(tmpRect);

               IF ColorFlag THEN
                  PmForeColor(kBlack)
               ELSE
```

```
                BEGIN
                StuffHex(@tmpPat, 'FFFFFFFFFFFFFFFF');
                   PenPat(tmpPat);
                   StuffHex(@tmpPat, '0000000000000000');
                   BackPat(tmpPat);
                END;
             yn := (yTiles * ySize);
             xn := (xTiles * xSize);
             FOR count := 0 TO xTiles DO
                BEGIN
                   MoveTo(Edge, Edge + (count * ySize));
                   Line(xn, 0);
                END;
             FOR count := 0 TO yTiles DO
                BEGIN
                   MoveTo(Edge + (count * xSize), Edge);
                   Line(0, yn);
                END;

             TextFace([shadow]);
             FOR count := 1 TO numTiles DO
                BEGIN
                   GetCountRect(count, tmpRect);
                   DrawTile(Board[count], tmpRect)
                END;
             TextFace([]);
          END;
    END;

{Handles the Move.}
    PROCEDURE DoMove (T : integer);
       VAR
          CanMove, UnLegal : boolean;
          tmpRect : Rect;
          CT, temp, nn, Rw, Dir : integer;
          Buf : ARRAY[1..MaxTiles] OF integer;
    BEGIN
       WITH thePuzzle^^ DO
          BEGIN
             CanMove := false;
             Rw := ((T - 1) DIV xTiles);

             Dir := 0;
             REPEAT
                UnLegal := false;
                Dir := Dir + 1;
                nn := 0;
                CT := T;
                REPEAT
                   nn := nn + 1;
                   Buf[nn] := CT;
                   IF Board[CT] = numTiles THEN
                      BEGIN
                         IF nn = 1 THEN
                            UnLegal := true
                         ELSE
                            CanMove := true;
                      END
                   ELSE
                      DoDir(Dir, xTiles, numTiles,
UnLegal, CT);
                UNTIL CanMove OR UnLegal;
             UNTIL CanMove OR (Dir >= 4);

             IF CanMove THEN
                BEGIN
                   TextFace([shadow]);
                   FOR temp := nn DOWNTO 2 DO
                      BEGIN
                Board[Buf[temp]] := Board[Buf[temp - 1]];
                      GetCountRect(Buf[temp], tmpRect);
                      DrawTile(Board[Buf[temp]], tmpRect)
                      END;
                Board[Buf[1]] := numTiles;
                GetCountRect(Buf[1], tmpRect);
                DrawTile(Board[Buf[1]], tmpRect);
                TextFace([]);

                nn := 0;
                REPEAT
                   nn := nn + 1;
                   UnLegal := (Board[nn] <> nn);
                UNTIL (nn = numTiles) OR UnLegal;
                IF NOT UnLegal THEN
                   BEGIN
                   SetRect(tmpRect, 0, 0, xBig, yBig);
                      IF Normal THEN
                         temp := Flash
                      ELSE
                         temp := Flash DIV 2;
                      FOR nn := 1 TO temp DO
                         InvertRect(tmpRect);
                      SysBeep(1);
                   END;
             END;
          END;
    END;

{Save new Color settings to resource}
    PROCEDURE SaveColor (T : integer;
                         C : RGBColor);
       VAR
          RGBH : RGBHandle;
          temp : integer;
    BEGIN
       RGBH := POINTER(GetResource(ResT,
thePuzzle^^.MyID));
       IF RGBH <> NIL THEN
          BEGIN
             RGBH^^[T] := C;

             ChangedResource(handle(RGBH));
             WriteResource(handle(RGBH));
             ReleaseResource(handle(RGBH));
          END;
    END;

{Handles changing the Color.}
    PROCEDURE DoColor (T : integer);
       VAR
          P : point;
          tmpRect : Rect;
          N : integer;
          C, NewC : RGBColor;
    BEGIN
       WITH thePuzzle^^ DO
          IF (T >= 1) AND (T <= numTiles + 1) THEN
             BEGIN
                P.v := 0;
                P.h := 0;
                IF T = numTiles + 1 THEN
                   N := numTiles + 2
                ELSE
                   N := Board[T] + 1;
                GetEntryColor(myPal, N, C);
                IF GetColor(P, 'Pick the Color for this
tile:', C, NewC) THEN
                   BEGIN
                      SetEntryColor(myPal, N, NewC);
                      SaveColor(N, NewC);
                      ActivatePalette(PuzzleWindow);
                      SetRect(tmpRect, 0, 0, xBig, yBig);
                      InvalRect(tmpRect);
                   END;
             END;
    END;
```

```
{Is the Option key down?}
      FUNCTION OptDown : boolean;
      BEGIN
      OptDown := (BitAnd(optionKey, theEvent.modifiers) <> 0);
      END;

{Handles a Mouse Down.}
      PROCEDURE DoMouse (P : point);
         VAR
            R : Rect;
            C, N : integer;
      BEGIN
         WITH thePuzzle^^ DO
            BEGIN
               GlobalToLocal(P);
               SetRect(R, 0, 0, xBig, yBig);
               IF PtInRect(P, R) THEN
                  BEGIN
                     N := numTiles + 1;
                     C := 0;
                     REPEAT
                        C := C + 1;
                        GetCountRect(C, R);
                        IF PtInRect(P, R) THEN
                           N := C;
                     UNTIL (N <= numTiles) OR (C = numTiles);

                        IF OptDown THEN
                           BEGIN
                              IF ColorFlag THEN
                                 DoColor(N);
                           END
                        ELSE IF (N <= numTiles) THEN
                           DoMove(N);
                     END;
            END;
      END;

{Handles a Key Down (ie. Help).}
      PROCEDURE DoKey;
         VAR
            tempVirtual, v, dv, c : integer;
            tempChar : char;
            tempRect : rect;
            ll : longint;
            count : integer;
            tmpPat : Pattern;

         PROCEDURE ColorBurst (S : str255);
            VAR
               ch : char;
               nn : integer;
         BEGIN
            WITH thePuzzle^^ DO
               BEGIN
                  MoveTo((xBig - StringWidth(S)) DIV 2, v);
                  v := dv + v;
                  IF ColorFlag THEN
                     FOR nn := 1 TO Length(S) DO
                        BEGIN
                           ch := S[nn];
                           PmForeColor(C);
                           c := c + 1;
                           IF c = (numTiles + 2) THEN
                              C := 2;
                           DrawChar(ch);
                        END
                  ELSE
                     DrawString(S);
               END;
         END;

      BEGIN
         WITH thePuzzle^^ DO
            BEGIN
               tempChar := Chr(theEvent.message MOD 256);
            tempVirtual := (theEvent.message DIV 256) MOD 256;
               IF (tempVirtual = $72) OR (tempChar IN ['?',
'/', 'H', 'h']) THEN
                  BEGIN
                     SetRect(tempRect, 0, 0, xBig, yBig);
                     IF ColorFlag THEN
                        BEGIN
                           FOR count := 2 TO NumTiles DO
                              BEGIN
                                 PmBackColor(count);
                                 EraseRect(tempRect);
                              InsetRect(tempRect, 1, 1);
                              END;
                           PmBackColor(kWhite);
                           EraseRect(tempRect);
                        END
                     ELSE
                        BEGIN
                     StuffHex(@tmpPat, 'FFFFFFFFFFFFFFFF');
                           PenPat(tmpPat);
                     StuffHex(@tmpPat, '0000000000000000');
                           BackPat(tmpPat);
                           FOR count := 2 TO NumTiles DO
                              BEGIN
                              IF (count MOD 2 = 0) THEN
                                 PaintRect(tempRect)
                              ELSE
                                 EraseRect(tempRect);
                              InsetRect(tempRect, 1, 1);
                              END;
                           EraseRect(tempRect);
                        END;

                     dv := kLine;
                     v := tempRect.top + dv;
                     C := 2;

                     ColorBurst('Color Puzzle 1.0');
                     ColorBurst('by');
                     ColorBurst('Steve Sheets ');
                     ColorBurst('dedicated to');
                     ColorBurst('Fred Bockman');
                     ColorBurst('Steve Levinthal ');
                     Delay(HelpWait, ll);
                     SetRect(tempRect, 0, 0, xBig, yBig);
                     IF ColorFlag THEN
                        PmBackColor(kWhite);
                     EraseRect(tempRect);
                     InvalRect(tempRect);
                  END
            END;
      END;

   BEGIN
      Control := noErr;

      GetPort(oldPort);
      HLock(theDCE^.dCtlStorage);
      thePuzzle := PuzzleHandle(theDCE^.dCtlStorage);
      SetPort(thePuzzle^^.PuzzleWindow);

      CASE IOPB^.csCode OF
         accEvent :
   (We got an event.)
            BEGIN
               BlockMove(IOPB^.ioMisc, @theEvent,
SizeOf(theEvent));
               CASE theEvent.what OF
                  updateEvt :
```


© The Definitive MacTutor, Vol. 4

```
                    BEGIN
                        BeginUpdate(WindowPtr(theEvent.message));
                        SetPort(WindowPtr(theEvent.message));
                        DrawBoard;
                        EndUpdate(WindowPtr(theEvent.message));
                    END;
                mouseDown :
                    IF FindWindow(theEvent.where, awindow)
IN [inSysWindow, inContent] THEN
                        IF aWindow =
thePuzzle^^.PuzzleWindow THEN
                            DoMouse(theEvent.where);
                keyDown, autoKey :
                    DoKey;
                OTHERWISE
            END;      {End of event CASE statement}
        END;
    accUndo, accCut, accCopy, accPaste, accClear :
        SysBeep(1);
    OTHERWISE
  END;

  HUnLock(Handle(thePuzzle));
  SetPort(oldPort);
END;

{DA's donothing Status Routine.}
FUNCTION Status (theDCE : DCtlPtr;
                 IOPB : ParmBlkPtr) : OSErr;
BEGIN
  Status := noErr;
END;

{DA's Close Routine: Dispose of the Palette, the Window Ptr &
the}
{  Data Handle.}
FUNCTION Close (theDCE : DCtlPtr;
                IOPB : ParmBlkPtr) : OSErr;
  VAR
    thePuzzle : PuzzleHandle;
BEGIN
  Close := noErr;

  HLock(theDCE^.dCtlStorage);
  thePuzzle := PuzzleHandle(theDCE^.dCtlStorage);
  WITH thePuzzle^^ DO
    BEGIN
      IF myPal <> NIL THEN
        DisposePalette(myPal);
      DisposeWindow(PuzzleWindow);
    END;
  HUnLock(theDCE^.dCtlStorage);

  IF theDCE^.dCtlRefNum < 0 THEN
    HPurge(Handle(theDCE^.dCtlDriver));

  theDCE^.dCtlWindow := NIL;
  DisposHandle(theDCE^.dCtlStorage);
  theDCE^.dCtlStorage := NIL;
END;

{Standard LSP Desk Accessory Main Routine}
FUNCTION Main;
BEGIN
  CASE driveCall OF
    DriverOpen :
      Main := Open(theDCE);
    DriverPrime :
      Main := Prime(theDCE, IOPB);
    DriverControl :
      BEGIN
        BitClr(@theDCE^.dCtlFlags, 15 - dCtlEnable);
        Main := Control(theDCE, IOPB);
        BitSet(@theDCE^.dCtlFlags, 15 - dCtlEnable);
      END;
    DriverStatus :
      Main := Status(theDCE, IOPB);
    DriverClose :
      Main := Close(theDCE, IOPB);
  END;
END;

END.
/*----------------------

    ColorPuzzle.r  -  Resources for the Color Puzzle DA
     used to store the Palette RGB values between boots.

    Steve Sheets 2/3/88
    All rights reserved.

----------------------*/

#include "Types.r"

data 'RGBs' (-16000) {
   $"43D8 95AE FFFF FFFF 8659 9785 4ECA 8BFF"
   $"06E2 FFFF FFFF 0000 FFFF 0000 FFFF 0000"
   $"FFFF FFFF FFFF 04E9 0EAE 0000 0000 FFFF"
   $"0000 FFFF 0000 FFFF 80C7 061A FFFF 0000"
   $"FFFF 22D4 827A D4FF FFFF 24D3 77A3 A426"
   $"60C4 FFFF C0FF 6091 0001 8888 8888 8888"
   $"5282 2638 FFFF"
};

data 'RGBs' (-15999) {
   $"A5B9 8160 07AB FFFF 038B 0580 FFFF 04D5"
   $"8295 FFFF 0001 FD8F FFFF 051B FFFF 882A"
   $"0961 FFFF 0854 828F FFFF 051F FB98 FFFF"
   $"075A FFFF 7CF0 051B FFFF 0282 0768 FFFF"
   $"023D 8533 FFFF 046E FAB5 8666 0CCD FFFF"
   $"FFFF 68F6 68F6 FFFF 6ABA B7C3 FC9B 6556"
   $"FFFF ACFC 63C2 FFFF 6C10 668D FFFF 6316"
   $"B628 FFFF 63D7 FFFF FFFF 6556 FFFF B60E"
   $"6B90 FFFF 714E B0BE FFFF 6661 FFFF F73D"
   $"66A3 FFFF ADC3 668D 0062 E01F FFFF 8564"
   $"1979 FFFF FFFF 542D A5B9 8160 07AB FFFF"
   $"038B 0580 FFFF 04D5 8295 FFFF 0001 FD8F"
   $"FFFF 051B FFFF 882A 0961 FFFF 0854 828F"
   $"FFFF 051F FB98 FFFF 075A FFFF 7CF0 051B"
   $"FFFF 0282 0768 FFFF 023D 8533 FFFF 046E"
   $"FAB5 8666 0CCD FFFF FFFF 68F6 68F6 FFFF"
   $"6ABA B7C3 FC9B 6556 FFFF ACFC 63C2 FFFF"
   $"6C10 668D FFFF 6316 B628 FFFF 63D7 FFFF"
   $"FFFF 6556 FFFF B60E 6B90 FFFF 714E B0BE"
   $"FFFF 6661 FFFF F73D 66A3 FFFF ADC3 668D"
   $"0062 E01F FFFF 8564 1979 FFFF FFFF 542D"
   $"A5B9 8160 07AB FFFF 038B 0580 FFFF 04D5"
   $"8295 FFFF 0001 FD8F FFFF 051B FFFF 882A"
   $"0961 FFFF 0854 828F FFFF 051F FB98 FFFF"
   $"075A FFFF 7CF0 051B FFFF 0282 0768 FFFF"
   $"023D 8533 FFFF 046E FAB5 8666 0CCD FFFF"
   $"FFFF 68F6 68F6 FFFF 6ABA B7C3 FC9B 6556"
   $"FFFF ACFC 63C2 FFFF 6C10 668D FFFF 6316"
   $"B628 FFFF 63D7 FFFF FFFF 6556 FFFF B60E"
   $"6B90 FFFF 714E B0BE FFFF 6661 FFFF F73D"
   $"66A3 FFFF ADC3 668D FFFF FFFF FFFF FFFF"
   $"FFFF FFFF FFFF FFFF FFFF 4482 B1D9 0000"
   $"0000 FFFF 0000 FFFF 0000 FFFF FFFF 0000"
   $"FFFF 0000 FFFF 0000 FFFF FFFF FFFF 0000"
   $"0000 0000 0000 FFFF 0000 FFFF 0000 FFFF"
   $"FFFF 0000 FFFF 0000 FFFF 0000 FFFF FFFF"
   $"FFFF 0000 0000 0000 0000 FFFF 0000 FFFF"
   $"0000 8888 8888 8888 1303 9828 FFFF"
};
```

Macintosh II
Palette Animation

Steven C. Sheets
Contributing Editor
MacTutor Vol. 4 No. 5

PalFun: an Example of Palette Manipulation

Last month's column explained the basics of the Palette Manager, the new Toolbox calls designed to handle the color environment of the Mac //. To fully understand this month's column, the reader will have to have either read the previous article and/or have worked with the Palette Manager on his own. Hopefully any new ideas/examples will be explained this month, but the reader must have a certain level of knowledge of Macintosh // Color to start with (this author hates 2-part TV Shows which spend most of the second part rehashing the events of the first half). This month's article will explain the idea of Palette Animation, or how to directly manipulate colors in the Palette to change shapes on the Macintosh // screen. The first portion of this article will describe the concepts involved in Palette Animation. The second section will describe special effect techniques that can be used with Palette Animation. The last section is an explanation of this month's sample program, PalFun.

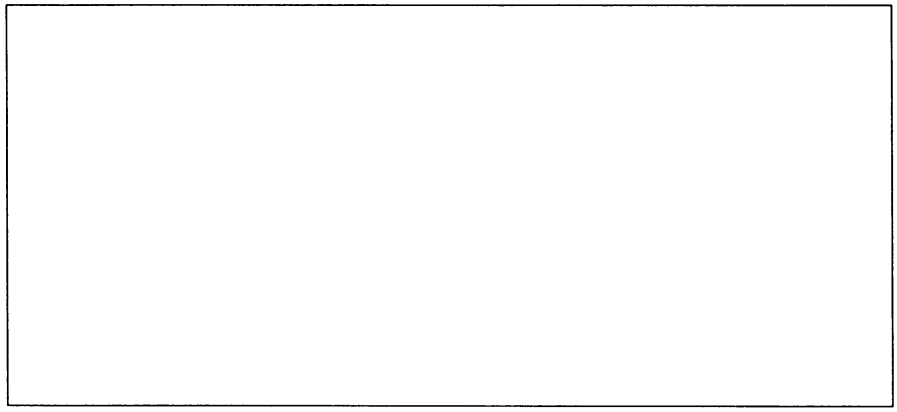


Fig. 1 Rainbow function animates color palette

Palette Animation: Changing the Color Environment

Imagine a sample Macintosh // screen, using the information described last month. The color screen has several windows on it, each one having a Palette and a group of colors that it wants to draw with. The top most window gets first choice of colors. Based on how many colors it wants and the tolerance of each color (ie. how picky it is in finding a closely matching color), the Palette Manager has changed certain colors in the CLUT (Color Look Up Table) of the video card. If there are any entries in the CLUT unused (ie. not reserved by someone), the next lower Window gets it's choice of Colors, and so on, and so on.

At some point, the number of available entries on the video card will run out. If there are any windows with Palette that have not gotten a chance to allocate entries, they will just have to make do (ie. use the given colors in the color environment). If there are several windows, this may happen to the bottom most window, or it may happen to the top most window if there are not very many entries in the CLUT of the video card.

Back to imagining the sample Macintosh // screen with multiple windows. Imagine that each one has a certain shade of Red it wants to display. The first window get it's exact Red $FFFF,$0,$0 (ie. the Palette Manager changes the CLUT of the video card so that some entry on it has this value). The second window happens to need the exact shade of Red $FFFF,$0,$0. Since the second window is more than happy to use the shade allocated by the first window, no entries are use by the second window to describe this red. The third window wants it's own shade of Red $CCCC,$0,$0, so it is allocated a color entry (there are still enough left). Finally the last window wants a Red shade of $EEEE,$0,$0, but there are no color entries left unallocated, so it does not change the Color environment.

When each Window tries to draw it's own shade of Red, it has to use the Colors available on the card (assume no other shades of Red on the video card in this example). Windows one and two use the $FFFF,$0,$0 shade of red, while Window three uses the $CCCC,$0,$0 shade. When window four tries to draw it's shade, Color Quickdraw will probably pick $FFFF,$0,$0 to color with. The selection of which color is actually going to be drawn with is transparent to the programmer; Color Quickdraw calculates the best match, not the program.

Remember that the CLUT contains the information of what colors are actually being displayed on the screen. The video card uses a RAM buffer to model the screen. Each pixel on the screen matches a value in the buffer. The buffer does not have a RGB value for each pixel, instead it has a pixel value. The size (in bits) of this value is fixed at any given moment, but can be changed depending on how much memory is allocated to the buffer (done by the Monitor portion of the Control Panel). So called pixel values in the buffer correspond to an entry in the video card CLUT. Thus a green pixel on the screen might have the pixel value of 7, which would match a color entry in the CLUT which has a pixel value of 7 and a RGB value of $000, $FFFF, $0000.

© The Definitive MacTutor, Vol. 4

This matching of the video buffer pixel values to the CLUT pixel values tells the card which colors to display on the screen.

Now things suddenly change. The program decided that window one really wants to draw using a shade of Blue $0,$0,$FFFF. The program uses SetEntryColor and ActivatePalette to change Red to Blue. In order for the Window to get this color, the Palette Manager needs to add a Blue entry to the CLUT (assume again there are no Blues in the current CLUT). The exact work of which colors stay and which change is dependent on Tolerances and other variables, but for this example, the Red $FFFF,$0,$0 entry in the CLUT is change to Blue $0,$0,$FFFF. Notice none of the Pixel values in the video buffer have changed. The RGB value of a certain entry in the CLUT has changed, and nothing else.

Back to the screen, while the user watches, everywhere were there was the Red shade $FFFF,$0,$0, there is now a Blue shade $0,$0,$FFFF. Through out the environment, all windows that showed that exact shade of Red, will show Blue. Red lips come out Blue, not exactly what the artist originally wanted. The video card can not change all the former Red pixel values in the video buffer; it has no idea what RGB colors each window wants to display.

This is why when each Palette is connect to a window using SetPalette, there is a cUpdate Boolean that needs to be passed. If cUpdate is true, the window will automatically be updated whenever the color environment changes. Then each window can redraw to the best color using the color environment. The flag should almost always be set to true. Again, this updating is completely transparent to the programmer; there is no code in the main program that causes updates whenever the color environment is changed.

Back at the windows, the top most window is updated. It draws using Blue. The lower three windows draw with the closest shade of Red, $CCCC,$0,$0. Now the second window is selected and becomes the top most window. The program does not change the Palettes, but because the top most window has changed, the color environment is about to. The new top most window has the highest priority when allocating entries in the CLUT, so Red $FFFF,$0,$0 is back on the table. The former top most window still gets it's Blue (due to the number of entries in the CLUT), so the CLUT is changed. Again all the windows most be updated in order for them to show the best possible colors.

The above model of changing the color environment is important to understand. The reader should have a feel for what happens when the color environment changes before going on.

Palette Animation: Animating Colors

Now think about what the Palette Manager did when it changed the color environment. It changed a few bytes of information in the CLUT. Because of this, *no matter how many colors or where there were, those colors on the screen suddenly changed.* To do the same thing using Copybits would be many factors slower. Accessing memory on Nubus cards is slower than normal memory. Even if access time was not slow, every pixel with the old color must be changed. If large areas of the screen are that color, redrawing pixels is an extremely slow task. This is assuming that the program knew where all the pixels of a certain color were to begin with. Palette Animation simply consists of changing colors on the screen to create animation effect by manipulating the CLUT, and not the pixel values.

In the above model, while colors do get changed, actual animation would be slow. Every time the environment changed, all the window are updated. Even if the animating window did not have it's cUpdate flag set, other windows on the screen would need to be updated. It would be faster to use Copybits directly to create animation this way. Remember all Palette colors in the above model were Tolerant or Courteous Colors. All normal Palette use these types of colors, because when Tolerant and Courteous Colors are allocated on the CLUT, they can be used by other windows.

There is another type of Color Entry: Animating Color. When these Colors are added to a CLUT, they are for exclusive use of the window/palette that allocated them. If a window was to be drawn using strictly animation colors, the colors of the entries in it's palette could be changed without effect the colors of the rest of the windows. Those other windows will never share animation colors with the animated window.

Animating Colors are created the same way as Tolerant or Courteous colors. A Palette containing Animating Colors can be created with NewPalette or GetNewPalette, and attached to a window with SetPalette. Color Entries, which consist of RGB value, Usage value and Tolerance value, can be set with SetEntryColor and SetEntryUsage. Animating Colors have a usage of 4, while the Tolerance is ignored.

To draw with an Animating Color, PmForeColor and PmBackColor must be used. If RGBForeColor or RGBBackColor is used to set the Foreground or Background Color on an Animating Window, that color will NOT be drawn using Animating Colors. Instead it will use the best available color in the color environment.

To change the animating Colors (and thus the screen colors) without changing the color environment, there are two special Palette Manager calls. AnimateEntry and AnimatePalette are designed to do this type of palette animation. Both calls are designed to work strictly with Animating Colors in a Palette. They change RGB values of certain entries of the Palette. The RGB values of entries in the CLUT of the video card will then be changed, causing the actual change of color on the screen. Since no other windows have the animating color, the color environment is not considered to have changed, so no update events are posted.

AnimateEntry is passed the animating window, the entry number of the Animating Color that will be changed, and the new RGB Color it will be changed to. AnimatePalette is passed the animating window, a Color Table (containing numerous RGB values), an Index value, Destination value and Length value. Beginning at the Index entry in the color table, a Length number of entries are copied to the Palette starting at the Destination entry. Thus, this call performs the same function as AnimateEn-

try, but for a group of animating colors.

Special Effect: Moving Colors

Now that the Palette Manager provides the capability of quickly changing the colors on the screen, how can a programmer use this to create animation effects? Consider what normal animation using normal Quickdraw requires. If a ball was suppose to be animated across the screen at a good speed, first the ball is drawn on on the screen. Then a certain length of time goes by to allow the user to see the ball. Next the ball is quickly erased, and redrawn at the new position. This may be repeated a number of times across the screen. To a viewer, it would look like the ball is being thrown across the screen.

To do the same thing with Palette animation, a Palette is created with 3 entries and attached to a window. (One entry for the background, two for the two positions of the ball). The background will be green, while the ball will be white, so entries zero and two (entries are counted from 0 to N) will be set to RGB Green $0,$FFFF,$0, while entry one will be White $FFFF,$FFFF,$FFFF.

Next the window is drawn (updated). The background is drawn to green, using PmBackColor, not RGBBackColor. The ball in the first position is drawn using entry one. The screen will show the white ball at this point. Next the second position of the ball is drawn using the third entry in the color table. Since that entry is set to green like the background, to the viewer, that ball is invisible.

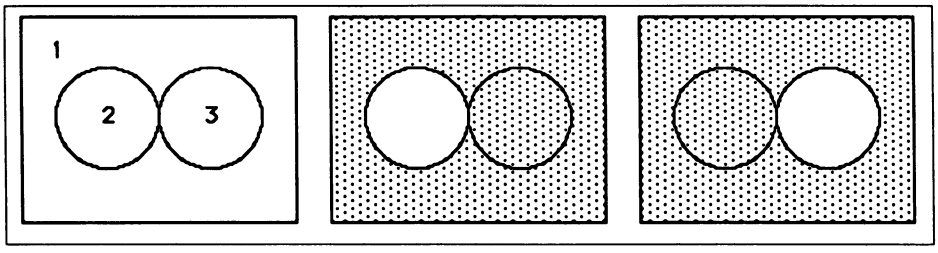


Fig. 2 Mixing colors

Now the animation starts. AnimateEntry is used to set entry one to green $0,$FFFF,$0 and entry two to white $FFFF,$FFFF,$FFFF. To the viewer, it looks like the ball has moved position. If there were more positions for the ball, there would be more entries (one for each new ball position). In that case, the animation routine would wait a certain amount of time so that the animation is not to fast for the viewer to see. Then ball is moved to the next position in the same manner (ie. entry two is set to green while entry three is set to white), and so on, and so on.

Notice that it would take the same amount of time to animate the Ball using Palette Animation no matter how big the ball is. Using normal animation techniques, speed of animation usually goes down for large objects. Also, the ball's position can be precalculated, something that may take too much time if done while animating.

Animation Speed is important for all types of animation. If a ball is suppose to move across the screen at a set speed, two concepts effect this movement speed: distance between images of the ball and the speed at which these images can be changed (sometimes measured in images or frames per second). If the speed of animation is to slow, the distance between the ball must be increased. However, large distances between objects causes jerky animation, and is noticeable by viewer. If the speed of animation is very slow, the screen will actually be refreshed while quickdraw is drawing in the video buffer. This causes a ripple effect that is a characteristic of bad animation! It is better to have small movement and a higher animation speed. It might seem that the ball must be moved at least it's own size between each image or palette animation will not work. Not so...

Special Effects: Overlaps

Again examine what happens during normal animation of a ball across the screen. The ball is drawn in white on green. After a time, the ball is erased (ie. drawn in green) and redrawn at the new position. If the new position overlaps the old position, there will be pixels on the screen that were white at the beginning and are still white at the end of this operation.

Therefore, the pixels in this example can be divided into four groups: those that always stay green, those that always stay white, those that start green, but change to white and those that start white, but change to green. Broken down that way, Palette Animation is simple. First a Palette is created with 4 Animating Color entries, the first and fourth are set to green, while the second and third are set to white. The background is drawn to green using the first animating color. The portion of the ball that is in the first position, but does not overlap the second is then drawn in white using the second color. The portion of the ball that is in the first position, and overlaps the second ball's position is then drawn in white using the third color. Finally the portion of the ball that is in the second position, but does not overlap the first is then drawn in green using the fourth color. To the user, it would appear that there is only one ball in the first position.

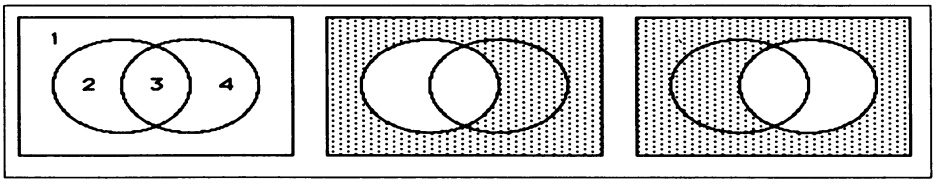


Fig. 3 Ball Animation

To animate, the second color, which was white, is set to green, while the fourth color which was green, is set to white. The

first and third colors would stay the same. For the user, it appears as if the ball has moved to the second position.

This is the basis of Palette Animation. If the ball was expanded so that it's colors were Red, White and Blue, while the background was changed to show Green, Orange and Purple, the picture can still be broken down into groups of pixels that change from one color to another between two frames. First, determine the pixels that never change, then determine the pixels that change from one color to another. This concept can be expanded for more frames and/or more objects. In all cases, group the pixels into sets which change the same colors from one frame to another.

The above techniques can be used for numerous effects. Many arcade games use these techniques along with normal animation. The twinkling of a rocket's jet or movement of an alien's fangs are both simple two step Palette Animation. This can be done separately while the rest of the game is being animated using traditional methods. It can be done no matter where the rocket is or how many aliens there are. Due to the low overhead in doing this type of animation, some graphic adventure games prefer simple animation using Palette animation. The twitching of an opponents eyes or the movement of an ocean wave are some common effects of such games. The effects are only limited by the programmer's imagination.

Special Effects: Rainbows and Fades

Two other types of Special Effects using Palette Animation should be mentioned: Fades and Rainbows. An image drawn with Animating Colors can be easily Faded into some neutral Color. This is an attractive way to exit from an image (ie. similar to the way Movies dissolve from one scene to another). To do this, first a set number of intermediate steps is chosen to work with. Then a loop is cycled for that number of times. In that loop, each entry in the Palette is changed a constant amount so that when the entire loop is done, all the entries will have the same RGB value. For example, to fade to black in 256 cycles, during each cycle each entry's RGB values are reduced by 256th of the original value. To the viewer, all the colors will slowly fade to Black. An entry which starts with the RGB value of $0,$FF00,$0 would be reduced by $0,$00FF,$0, while an entry with a RGB value of $8000,$6700,$0 would be reduced by a value of $0080,$0067,$0 each cycle. Notice each Red, Green and Blue part of an RGB value is calculated separately. Also an image can fade to any RGB value, not just White, Black or even Gray.

Once an image is faded out, a new image can be faded in with the same technique. Assuming the second image uses the same Palette, it can be drawn while the screen is at the neutral color. Since the colors are still all the same, the new image will be invisible. Then it can be faded to either the old values, or even to any new values. The entries do not care at which point the program stops adjusting them.

The other note-worthy effect is a Rainbow motion. A Palette is created which contains some spectrum of colors. While any arbitrary color set can be selected, in many cases, a Rainbow spectrum is chosen. Once the Palette is create, some shape is drawn so that the full range of entries are used. Different shapes include parallel lines, arcs in a circle, circles within each other or even lines in a web pattern. Then the Palette is animated so that every entry's RGB value is moved to the next entry (and the last one is moved to the front of the list). The resulting animated motion can be hypnotizing.

PalFun: A Dynamic Colorful Example

PalFun, this months sample program, demonstrates some of the techniques described this month, as well as some of the concepts discussed last issue. The reader can for the most part ignore the main portion of the program (an exceptionally simple program) once he understands that this is a shell designed to create, display and animate Palette windows. In PalFun, every window has a routine associated with it that creates the Window/Palette (MakeSomeWindow), a routine associated that updates the image (UpdateSomeWindow) and some have a routine associated that animates the image (AnimateSomeWindow).

Red Window, Green Window and Blue Window demonstrate various ways to create normal Palettes. Try picking different windows, and notice how this effects the current color environment. Also, notice how each window is automatically updated. Change the bit depth of the Video Card (ie. number of bits per pixel) and see how the Windows change. Notice that the Blue window uses a method of Color Priority (mentioned last month) that allows it to appear at it's best no matter what Pixel or Window depth.

Current Color Window uses Explicit Colors (explained last month). Explicit colors provide a convenient way for the user to view the current color environment. An Explicit entry is not drawn using it's RGB value. Instead it displays the CLUT's color at it's corresponding position. For example, the fifth entry in Palette will display the fifth entry in the CLUT of the video card. Explicit colors is the only method to view Animating Colors in some window other than the one the Animating Colors were created for. When selecting various windows and/or running through Animation techniques, keep an eye on the Current Color Window to view what's happening. Converting Current Color into a Desk Accessory creates a useful tool for anyone who tinkers with the Palettes.

Ball Window gives the basic example of moving a non-overlapping ball around a screen. Shapes demonstrates how to calculate the overlaps and pixels of different, completely arbitrary 2-bit images in order to step through the images. It uses unions and intersections of regions in order to find out what pixels change into what colors. For example, given three regions (images), the pixels that will never change colors are the ones that do not intersect the 3 regions, or there intersections between A region, B region and C region. The pixels that would be set (Black) for the first image, then clear (White) for the next 2 images are the pixels that in A Region minus the ones that are in B and C regions. Notice that if the result of any such calculation is the empty region, there is no pixels of that color and that pixel's

Color Entry really does not have to be allocated/used. Fade and Rainbow are the last two windows. They demonstrate the above mentioned Fade and Rainbow techniques.

Last Comments: Palette Techniques

Here are some last comments for this month:

- Always start a Palette with a White and Black color entry. Due to a bug in the Palette Manager, the first entry must be White or Black. Even without this, it is good practice to always have White and Black as the first two colors in a Palette (remember Color Priorities?).

- Animating Windows should still have the cUpdates flag set when SetPalette is called. This will prevent undesired animation when ActivatePalette reserves entries that were being used by the animating window.

- When using the Palette Manager, the user may sometimes see the Background Desktop pattern change colors when the windows are dragged around. This happens most often when the Colors used in the Background Pattern (set by the Control Panel) are something other than Black and White. What has occurred is that the Palette Manager has changed the Color Environment, but the Window Manager (which takes care of the Background) does not known this. This is due to a problem with the Window Manager that will hopefully be fixed.

- While Shape demonstrates how to calculate animating arbitrary images, it is best to design the image with Palette animation in mind. Keep the colors in separate portions of the screen, so they do not intersect too many times and need separate entries. The lower the number of sets of pixels, the more images that can be animated. A good Palette Manager example would be a program that takes arbitrary MacPaint documents and displays them one after another using Palette Animation. The Application could be smart enough to keep track of Pixel differences (ie. unions and intersects of bitmaps) so that it keeps track of the maximum number of entries that need to be used. The Application could keep adding MacPaint pictures until the maximum number of entries of the Video Card is reached, and then quickly flash through the pictures.

- When should a programmer use SetPalette vs using a 'pltt' resource? This is the same as asking when a programmer should use NewWindow vs using a 'WIND' resource; it depends on when the settings of the Window or Palette are calculated. If they are calculated at run time or constantly changing during the development of the program, use SetPalette. This provides the most flexibility to the programmer. If the Palette is precalculated (ie. always the same for every run of the program), use the 'pltt' resource to separate the Color from the Code. This is done for the same reason most programmers separate the Text from the Code: Localization.

- A Special Thank you to Art Cabral for the help he provided in understand the Palette Manager (No more 2 minute long phone calls, Art).

Next issue, manipulating Color Icons and a Color Icon Editor. Even if there are tools to do this by then, the understanding of advance Color Quickdraw data structures is worth the study.

About the Program Code

PalFun is written in MPW Pascal, available on the MacTutor Source Code disk for this issue. The listing here has been translated into LS Pascal (and should work as is in Turbo Pascal also) by the Editor. The resources are also listed here in RMaker format, but the Rez format is on the Source Code Disk. The MPW version is a single program, but for LS Pascal, the program had to be broken into this main program, and two units: one unit for the globals and the other unit for all the Palette specific routines called PalFunStuff. Since LS Pascal is the most generic of all the Pascal compilers, this listing should be most easily entered into all the other compilers, MPW, Turbo and TML by just changing the USES statement.

```
{  PalFun by Steve Sheets 3/88   }
{Palette Manager Sample Program designed for MacTutor.}
{It demonstrate various Palette Animation effects.}

PROGRAM PalFun;

   USES
      ROM85, ColorQuickDraw, ColorWindowMgr, PaletteMgr,
PickerIntf, PalFunGlobals, PalFunStuff;

{********* Main Portion Programs ***********}

   PROCEDURE crash;
   BEGIN
      ExitToShell;
   END;

   PROCEDURE SetUp;
      VAR
         count : integer;
   BEGIN     {Standard Mac Program setup}
      InitGraf(@thePort);
      InitFonts;
      FlushEvents(everyEvent, 0);
      InitWindows;
      InitMenus;
      TEInit;
      InitDialogs(@crash);
      InitCursor;

      FOR count := appleM TO menuCount DO
         myMenus[count] := GetMenu(count);
      AddResMenu(myMenus[appleM], 'DRVR');
      FOR count := appleM TO menuCount DO
```

```
        BEGIN
            moveHHi(handle(myMenus[count]));
            HLock(handle(myMenus[count]));
        END;
    FOR count := appleM TO menuCount DO
        InsertMenu(myMenus[count], 0);
    DrawMenuBar;

    WITH screenBits.bounds DO
        SetRect(dragRect, 4, 24, right - 4, bottom - 4);
    doneFlag := FALSE;

    FOR count := 1 TO numWindows DO
        BEGIN
            MyWindow[count] := NIL;
            MyPalette[count] := NIL;
        END;
END;

{Given a Window ID number, close Window/Palette}
PROCEDURE CloseIt (N : INTEGER);
BEGIN
    IF (N > 0) AND (N <= numWindows) THEN
        BEGIN
            IF MyPalette[N] <> NIL THEN
                DisposePalette(MyPalette[N]);
            MyPalette[N] := NIL;
            IF MyWindow[N] <> NIL THEN
                DisposeWindow(MyWindow[N]);
            MyWindow[N] := NIL;
        END;
END;

{Gets the Front most Window ID number.}
FUNCTION GetWindowNum (W : WindowPtr) : INTEGER;
    VAR
        N, count : INTEGER;
BEGIN
    IF W = NIL THEN
        GetWindowNum := 0
    ELSE
        BEGIN
            N := 0;
            FOR count := 1 TO numWindows DO
                IF MyWindow[count] = W THEN
                    N := count;
            GetWindowNum := N;
        END;
END;

{ Standard Handling of the Menu. Selecting the Window Menu,
bring that window to the front and that's all (Palette Manager
handles changing colors and creating update). Animate Menu
animate the front window if it can. }
PROCEDURE DoCommand (mResult : LONGINT);
    VAR
        theItem : INTEGER;
        theMenu : INTEGER;
        name : Str255;
        N : INTEGER;
        dummy : Boolean;
        tempPort : GrafPtr;
BEGIN
    theItem := LoWord(mResult);
    theMenu := HiWord(mResult);
    CASE theMenu OF
        appleM :
            IF theItem = 1 THEN
                theItem := Alert(AlertID, NIL)
            ELSE
                BEGIN
                    GetItem(myMenus[appleM], theItem, name);
                    N := OpenDeskAcc(name);
                END;
        fileM :
            CASE theItem OF
                1 :
                    BEGIN
                        GetPort(tempPort);
                        SetPort(FrontWindow);
                        CASE GetWindowNum(FrontWindow) OF
                            ballW :
                                AnimBall;
                            shapeW :
                                AnimShape;
                            rainbowW :
                                AnimRainbow;
                            fadeW :
                                AnimFade;
                            OTHERWISE
                        END;
                        SetPort(tempPort);
                    END;
                2 :
                    CloseIt(GetWindowNum(FrontWindow));
                4 :
                    doneFlag := TRUE;
                OTHERWISE
            END;
        editM :
            dummy := SystemEdit(theItem - 1);
        winM :
            IF (theItem > 0) AND (theItem <= numWindows)
THEN
                BEGIN
                    IF MyWindow[theItem] = NIL THEN
                        BEGIN
                            CASE theItem OF
                                redW :
                                    MakeRed;
                                greenW :
                                    MakeGreen;
                                blueW :
                                    MakeBlue;
                                ballW :
                                    MakeBall;
                                curW :
                                    MakeCur;
                                shapeW :
                                    MakeShape;
                                rainbowW :
                                    MakeRainbow;
                                fadeW :
                                    MakeFade;
                                OTHERWISE
                            END;
                        END
                    ELSE
                        SelectWindow(MyWindow[theItem]);
                END;
            OTHERWISE
        END;
        HiliteMenu(0);
END;

{Extremely Standard Main Program Loop.}
PROCEDURE DoMainLoop;
    VAR
        theChar : CHAR;
        myEvent : EventRecord;
        whichWindow : WindowPtr;
        oldPort : GrafPtr;
        dummy : boolean;
BEGIN
    REPEAT
        SystemTask;
```

```
        IF GetNextEvent(everyEvent, myEvent) THEN
        CASE myEvent.what OF
        mouseDown :
           CASE FindWindow(myEvent.where, whichWindow) OF
              inSysWindow :
                 SystemClick(myEvent, whichWindow);
              inMenuBar :
                 DoCommand(MenuSelect(myEvent.where));
              inGoAway :
                 IF TrackGoAway(whichWindow, myEvent.where)
THEN
                 CloseIt(GetWindowNum(whichWindow));
              inDrag :
                 IF (FrontWindow <> whichWindow) THEN
                    SelectWindow(whichWindow)
                 ELSE
                    DragWindow(whichWindow, myEvent.where,
dragRect);
              inContent :
                 IF (FrontWindow <> whichWindow) THEN
                    SelectWindow(whichWindow);
           OTHERWISE
           END; {of mouseDown}
        keyDown, autoKey :
           BEGIN
              theChar := CHR(BitAnd(myEvent.message,
charCodeMask));
              IF BitAnd(myEvent.modifiers, cmdKey) <> 0
THEN
                 DoCommand(MenuKey(theChar));
           END;
        updateEvt :
           BEGIN
              whichWindow := WindowPtr(myEvent.message);
              IF whichWindow <> NIL THEN
                 BEGIN
                    GetPort(oldPort);
                    SetPort(whichWindow);
                    BeginUpdate(whichWindow);
                    CASE GetWindowNum(whichWindow) OF
                       redW :
                          DoRedUpdate;
                       greenW :
                          DoGreenUpdate;
                       blueW :
                          DoBlueUpdate;
                       curW :
                          DoCurUpdate;
                       ballW :
                          DoBallUpdate;
                       shapeW :
                          DoShapeUpdate;
                       rainbowW :
                          DoRainbowUpdate;
                       fadeW :
                          DoFadeUpdate;
                    OTHERWISE
                    END;
                    EndUpdate(whichWindow);
                    SetPort(oldPort);
                 END;
           END;
        OTHERWISE
        END;
     UNTIL doneFlag;
  END;

{Dispose of all the Palettes and closes all the Windows.}
  PROCEDURE CloseDown;
     VAR
        count : integer;
  BEGIN
     FOR count := 1 TO numWindows DO
        CloseIt(count);
     FOR count := appleM TO menuCount DO
        BEGIN
           DeleteMenu(count);
           DisposeMenu(myMenus[count]);
        END;
     DrawMenuBar;
  END;

{ Main Body Program.  Setup, Do it, Close down.}

BEGIN
  IF ColorQDExists THEN
     BEGIN
        SetUp;
        DoMainLoop;
        CloseDown;
     END;
END.
```

```
UNIT PalFunStuff;

INTERFACE

  USES
     ROM85, ColorQuickDraw, ColorWindowMgr, PaletteMgr,
PickerIntf, PalFunGlobals;

  FUNCTION ColorQDExists : boolean;

  PROCEDURE AnimBall;
  PROCEDURE AnimShape;
  PROCEDURE AnimRainbow;
  PROCEDURE AnimFade;

  PROCEDURE MakeRed;
  PROCEDURE MakeGreen;
  PROCEDURE MakeBlue;
  PROCEDURE MakeBall;
  PROCEDURE MakeCur;
  PROCEDURE MakeShape;
  PROCEDURE MakeRainbow;
  PROCEDURE MakeFade;

  PROCEDURE DoRedUpdate;
  PROCEDURE DoGreenUpdate;
  PROCEDURE DoBlueUpdate;
  PROCEDURE DoCurUpdate;
  PROCEDURE DoBallUpdate;
  PROCEDURE DoShapeUpdate;
  PROCEDURE DoRainbowUpdate;
  PROCEDURE DoFadeUpdate;

IMPLEMENTATION


{******************** General Tools ********************}

{Returns true if the Mac had Color Quickdraw.}
  FUNCTION ColorQDExists; {boolean}
     CONST
        ROM85Loc = $28E;
        TwoHighMask = $C000;
     TYPE
        WordPtr = ^INTEGER;
     VAR
        Wd : WordPtr;
  BEGIN
     Wd := POINTER(ROM85Loc);
     ColorQDExists := (BitAnd(Wd^, TwoHighMask) = 0);
  END;
```


© The Definitive MacTutor, Vol. 4

```
{Stuffs Red, Green & Blue into RGBColor}
   PROCEDURE SetRGB (VAR RGB : RGBColor;
                             R, G, B : INTEGER);
   BEGIN
      RGB.Red := R;
      RGB.Green := G;
      RGB.Blue := B;
   END;

{Copies RGBColor into RGBColor}
   PROCEDURE CopyRGB (RGBsrc : RGBColor;
                           VAR RGBdest : RGBColor);
   BEGIN
      RGBdest.Red := RGBsrc.Red;
      RGBdest.Green := RGBsrc.Green;
      RGBdest.Blue := RGBsrc.Blue;
   END;

{Delays a set length time.  usually until}
{ the screen in refreshed (prevents ripples)}
   PROCEDURE DoDelay (N : INTEGER);
      VAR
         L : LONGINT;
   BEGIN
      L := TickCount + N;
      WHILE L > TickCount DO
         ;
   END;

{Using 16 Bit Unsigned Integers: C:=A/B}
   PROCEDURE UnSignedDiv (A, B : INTEGER;
                           VAR C : INTEGER);
      VAR
         L : LongInt;
   BEGIN
      IF A <. 0 THEN
         L := A + 65536
      ELSE
         L := A;
      C := LoWord(L DIV B);
   END;

{Using 16 Bit Unsigned Integers: A:=A+B}
   PROCEDURE UnSignedAdd (VAR A : INTEGER;
                          B : INTEGER);
      VAR
         L : LongInt;
   BEGIN
      IF A < 0 THEN
         L := A + 65536 + B
      ELSE
         L := A + B;
      A := LoWord(L);
   END;

{Using 16 Bit Unsigned Integers: A:=A-B}
   PROCEDURE UnSignedSub (VAR A : INTEGER;
                          B : INTEGER);
      VAR
         L : LongInt;
   BEGIN
      IF A < 0 THEN
         L := A + 65536 - B
      ELSE
         L := A - B;
      A := integer(LoWord(L));
   END;

{********************* Color Table Tools *********************}

{Given number of Colors to be placed in it, creates a blank
CLUT.  Gives it}
{  an unique Seed and correct value, but no colors.}
   FUNCTION NewCT (N : integer) : CTabHandle;
      VAR
         MyCT : MyCTabHandle;
         count : integer;
   BEGIN
      MyCT := NIL;
      IF (N > 0) AND (N <= MaxCT) THEN
         BEGIN
            MyCT := POINTER(NewHandle((N * SIZEOF(ColorSpec))
+ (2 * SIZEOF(integer)) + SIZEOF(longint)));
            IF MyCT <> NIL THEN
               WITH MyCT^^ DO
                  BEGIN
                     ctSeed := GetCTSeed;
                     ctFlag := 0;
                     ctSize := N - 1;
                     FOR count := 0 TO N - 1 DO
                        WITH ctTable[count] DO
                           BEGIN
                              value := count;
                              SetRGB(rgb, 0, 0, 0);
                           END;
                  END;
         END;
      NewCT := POINTER(MyCT);
   END;

{Stuffs an RGB value in the Nth Color (numbered 0 to N) of the
CLUT.}
   PROCEDURE SetCTEntry (C : CTabHandle;
                              N, R, G, B : INTEGER);
      VAR
         MyCT : MyCTabHandle;
   BEGIN
      MyCT := POINTER(C);
      SetRGB(MyCT^^.ctTable[n].rgb, R, G, B);
   END;

{********************* Red *********************}

{Red Window displays encompassing red-shaded circles.}
{This creates a 3-D Globe effect.}

{Create Red Window/Palette with NewPalette & SetEntryColor
commands.}
   PROCEDURE MakeRed;
      VAR
         tempRect : rect;
         tempRGB : RGBColor;
         S : str255;
         count : integer;
   BEGIN
      SetRect(tempRect, 20, 40, 320, 340);
      GetIndString(S, StrID, 1);
      MyWindow[redW] := NewCWindow(NIL, tempRect, S, true,
noGrowDocProc, POINTER(-1), true, 0);

      MyPalette[redW] := NewPalette(128, NIL, pmTolerant, 0);
      SetRGB(tempRGB, ColorStart, 0, 0);
      FOR count := 0 TO 127 DO
         BEGIN
            SetEntryColor(MyPalette[redW], count, tempRGB);
            UnSignedSub(tempRGB.red, ColorInc);
         END;

      SetPalette(MyWindow[redW], MyPalette[redW], true);
   END;

{Draw the Red Window using RGBForeColor.}
   PROCEDURE DoRedUpdate;
      VAR
         tempRect : rect;
         tempRGB : RGBColor;
```

```
        count : integer;
   BEGIN
      SetRect(tempRect, 22, 22, 278, 278);
      SetRGB(tempRGB, ColorStart, 0, 0);
      FOR count := 0 TO 127 DO
         BEGIN
            RGBForeColor(tempRGB);
            PaintOval(tempRect);
            InsetRect(tempRect, 1, 1);
            UnSignedSub(tempRGB.red, ColorInc);
         END;
   END;

{******************** Green ********************}

{Green Window displays a Green Globe.}

{Create Green Window/Palette with NewPalette command & CLUT
procedures.}
   PROCEDURE MakeGreen;
      VAR
         tempRect : rect;
         tempRGB : RGBColor;
         tempCT : CTabHandle;
         Col : INTEGER;
         S : str255;
         count : integer;
   BEGIN
      SetRect(tempRect, 40, 60, 340, 360);
      GetIndString(S, StrID, 2);
      MyWindow[greenW] := NewCWindow(NIL, tempRect, S, true,
noGrowDocProc, POINTER(-1), true, 0);

      tempCT := NewCT(128);
      Col := ColorStart;
      FOR count := 0 TO 127 DO
         BEGIN
            SetCTEntry(tempCT, count, 0, Col, 0);
            UnSignedSub(Col, ColorInc);
         END;
      MyPalette[greenW] := NewPalette(128, tempCT, pmTolerant,
0);
      DisposHandle(Handle(tempCT));

      SetPalette(MyWindow[greenW], MyPalette[greenW], true);
   END;

{Draw the Green Window using PmForeColor.}
   PROCEDURE DoGreenUpdate;
      VAR
         tempRect : rect;
         count : integer;
   BEGIN
      SetRect(tempRect, 22, 22, 278, 278);
      FOR count := 0 TO 127 DO
         BEGIN
            PmForeColor(count);
            PaintOval(tempRect);
            InsetRect(tempRect, 1, 1);
         END;
   END;

{******************** Blue ********************}

{Display a Blue Globe (like Green Window), but now the colors
}
{are set up for better displaying (ie. Color Priority).}

{Create Green Window/Palette with NewPalette command & CLUT
procedures.}

   PROCEDURE MakeBlue;
      VAR
         tempRect : rect;
         tempRGB : RGBColor;
         tempCT : CTabHandle;
         Col : INTEGER;
         S : str255;
         h, v : integer;
   BEGIN
      SetRect(tempRect, 60, 80, 360, 380);
      GetIndString(S, StrID, 3);
      MyWindow[blueW] := NewCWindow(NIL, tempRect, S, true,
noGrowDocProc, POINTER(-1), true, 0);

      tempCT := NewCT(128);
      Col := ColorStart;
      FOR h := 0 TO 15 DO
         FOR v := 0 TO 7 DO
            BEGIN
               SetCTEntry(tempCT, (v * 16) + h, 0, 0, Col);
               UnSignedSub(Col, ColorInc);
            END;
      MyPalette[blueW] := NewPalette(128, tempCT, pmTolerant,
0);
      DisposHandle(Handle(tempCT));

      SetPalette(MyWindow[blueW], MyPalette[blueW], true);
   END;

{Draw the Blue Window using RGBForeColor.}
   PROCEDURE DoBlueUpdate;
      VAR
         tempRect : rect;
         tempRGB : RGBColor;
         count : integer;
   BEGIN
      SetRect(tempRect, 22, 22, 278, 278);
      SetRGB(tempRGB, 0, 0, ColorStart);
      FOR count := 0 TO 127 DO
         BEGIN
            RGBForeColor(tempRGB);
            PaintOval(tempRect);
            InsetRect(tempRect, 1, 1);
            UnSignedSub(tempRGB.Blue, ColorInc);
         END;
   END;

{******************** Current Color ********************}

{Displays the Current Color Enviroment}

{Create the current Color Window using Explicit colors }
{(Does not have to set the colors).}
   PROCEDURE MakeCur;
      VAR
         tempRect : rect;
         S : str255;
   BEGIN
      SetRect(tempRect, 100, 80, 420, 400);
      GetIndString(S, StrID, 4);
      MyWindow[curW] := NewCWindow(NIL, tempRect, S, true,
noGrowDocProc, POINTER(-1), true, 0);
      MyPalette[curW] := NewPalette(256, NIL, pmExplicit, 0);
      SetPalette(MyWindow[curW], MyPalette[curW], true);
   END;

{Draws the current Graphic Device Colors.}
   PROCEDURE DoCurUpdate;
      VAR
         x, y, n : integer;
         tempRect : rect;
   BEGIN
      n := 0;
      FOR y := 0 TO 15 DO
         FOR x := 0 TO 15 DO
```


© The Definitive MacTutor, Vol. 4

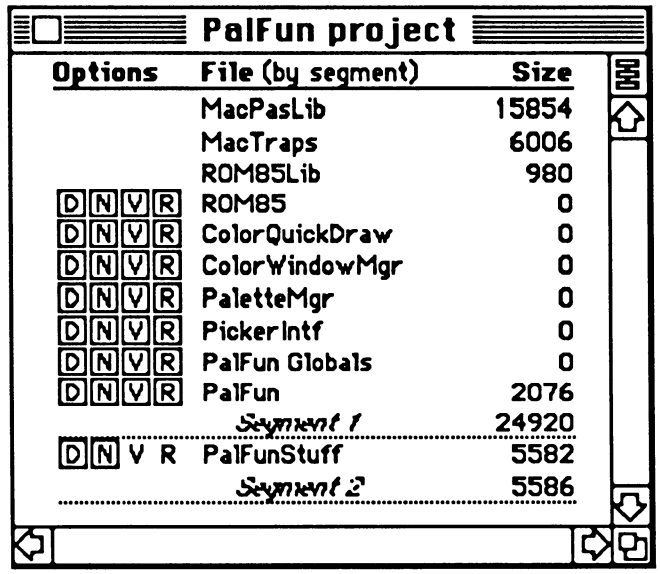
```
            BEGIN
               PmForeColor(n);
               SetRect(tempRect, x * 20, y * 20, (x + 1) *
20, (y + 1) * 20);
               PaintRect(tempRect);
               n := n + 1;
            END;
   END;

{******************** Ball ********************}

{Simple Palette Animation of a Ball Across the Screen}

{Create the Ball Animation Window using Animated colors.}
   PROCEDURE MakeBall;
      VAR
         tempRect : rect;
         tempRGB : RGBColor;
         S : str255;
         count : integer;
   BEGIN
      SetRect(tempRect, 100, 120, 400, 420);
      GetIndString(S, StrID, 5);
      MyWindow[ballW] := NewCWindow(NIL, tempRect, S, true,
noGrowDocProc, POINTER(-1), true, 0);
      MyPalette[ballW] := NewPalette(19, NIL, pmAnimated, 0);

      SetRGB(tempRGB, $FFFF, $FFFF, $FFFF);
      SetEntryColor(MyPalette[ballW], 0, tempRGB);
      SetRGB(tempRGB, 0, 0, 0);
      SetEntryColor(MyPalette[ballW], 1, tempRGB);
      tempRGB.blue := $FFFF;
      SetEntryColor(MyPalette[ballW], 2, tempRGB);
      tempRGB.blue := 0;
      tempRGB.red := $FFFF;
      FOR count := 3 TO 18 DO
         SetEntryColor(MyPalette[ballW], count, tempRGB);
      SetPalette(MyWindow[ballW], MyPalette[ballW], true);
   END;

{Draw the Balls in the window using PmForeColor.}
   PROCEDURE DoBallUpdate;
      VAR
         R : rect;
         count : integer;
   BEGIN
      SetRect(R, 0, 0, 10000, 10000);
      PmForeColor(18);
      PaintRect(R);

      FOR count := 2 TO 17 DO
         BEGIN
            R.top := 16 * count;
            R.left := R.top;
            R.bottom := R.top + 16;
            R.right := R.bottom;
            PmForeColor(count);
            PaintOval(R);
         END;
   END;

{Animate the Ball through the window using AnimateEntry.}
   PROCEDURE AnimBall;
      VAR
         R, B : RGBcolor;
         time, count, temp : integer;
   BEGIN
      SetRGB(R, $FFFF, 0, 0);
      SetRGB(B, 0, 0, $FFFF);
      FOR time := 1 TO 10 DO
         FOR count := 2 TO 17 DO
            BEGIN
               IF count = 17 THEN
                  temp := 2
               ELSE
                  temp := count + 1;
               DoDelay(1);
               AnimateEntry(MyWindow[ballW], count, R);
               AnimateEntry(MyWindow[ballW], temp, B);
            END;
   END;

{******************** Shape ********************}

{ Given 3 arbitrary regions (Black/White images), calculates
how to draw the window so that the images can be shuffled
through quickly. }

{Create the Shape Animation Window}
{('pltt' is automatically loaded in).}
   PROCEDURE MakeShape;
   BEGIN
      MyWindow[shapeW] := GetNewCWindow(ShapeID, NIL,
POINTER(-1));
   END;

{Draws Shape.  aRgn,bRgn,cRgn are the arbitrary images.}
   PROCEDURE DoShapeUpdate;
      VAR
         aRgn, bRgn, cRgn, TempRgn : RgnHandle;
         count : INTEGER;
         TempRect : Rect;
      PROCEDURE DrawTriag (h, v : INTEGER);
      BEGIN
         MoveTo(h + 25, v);
         Line(-25, 50);
         Line(50, 0);
         Line(-25, -50);
      END;
   BEGIN
      aRgn := NewRgn;
      OpenRgn;
      SetRect(tempRect, 10, 10, 60, 60);
      FrameOval(tempRect);
      SetRect(tempRect, 120, 10, 170, 60);
      FrameRect(tempRect);
      SetRect(tempRect, 120, 80, 170, 130);
      FrameRect(tempRect);
      SetRect(tempRect, 190, 10, 240, 60);
      FrameRect(tempRect);
      SetRect(tempRect, 190, 80, 240, 130);
      FrameRect(tempRect);
      SetRect(tempRect, 10, 80, 110, 81);
      FOR count := 1 TO 25 DO
         BEGIN
            FrameRect(tempRect);
            OffSetRect(tempRect, 0, 2);
         END;
      CloseRgn(aRgn);
      bRgn := NewRgn;
      OpenRgn;
      SetRect(tempRect, 35, 10, 85, 60);
      FrameOval(tempRect);
      SetRect(tempRect, 120, 10, 170, 60);
      FrameOval(tempRect);
      SetRect(tempRect, 120, 80, 170, 130);
      FrameOval(tempRect);
      SetRect(tempRect, 190, 10, 240, 60);
      FrameOval(tempRect);
      SetRect(tempRect, 190, 80, 240, 130);
      FrameOval(tempRect);
      SetRect(tempRect, 10, 80, 11, 130);
      FOR count := 1 TO 25 DO
         BEGIN
            FrameRect(tempRect);
            OffSetRect(tempRect, 4, 0);
```

```
        END;
      CloseRgn(bRgn);
      cRgn := NewRgn;
      OpenRgn;
      SetRect(tempRect, 60, 10, 110, 60);
      FrameOval(tempRect);
      DrawTriag(120, 10);
      DrawTriag(120, 80);
      DrawTriag(190, 10);
      DrawTriag(190, 80);
      MoveTo(60, 80);
      Line(50, 25);
      Line(-50, 25);
      Line(-50, -25);
      Line(50, -25);
      CloseRgn(cRgn);
      TempRgn := NewRgn;

{This Region will always be Red (Background)}
      PmForeColor(0);
      SetRect(tempRect, -32000, -32000, 32000, 32000);
      PaintRect(tempRect);

{This region will start Blue, change Red, stay Red}
      PmForeColor(1);
      DiffRgn(aRgn, bRgn, TempRgn);
      DiffRgn(TempRgn, cRgn, TempRgn);
      PaintRgn(TempRgn);

{This region will be Red,Blue,Red}
      PmForeColor(2);
      DiffRgn(bRgn, aRgn, TempRgn);
      DiffRgn(TempRgn, cRgn, TempRgn);
      PaintRgn(TempRgn);

{This region will be Blue,Blue,Red}
      PmForeColor(3);
      SectRgn(aRgn, bRgn, TempRgn);
      PaintRgn(TempRgn);

{This region will be Red,Red,Blue}
      PmForeColor(4);
      DiffRgn(cRgn, aRgn, TempRgn);
      DiffRgn(TempRgn, bRgn, TempRgn);
      PaintRgn(TempRgn);

{This region will be Blue,Red,Blue}
      PmForeColor(5);
      SectRgn(aRgn, cRgn, TempRgn);
      PaintRgn(TempRgn);

{This region will be Red,Blue,Blue}
      PmForeColor(6);
      SectRgn(bRgn, cRgn, TempRgn);
      PaintRgn(TempRgn);

{This Region will always be Blue}
      PmForeColor(7);
      SectRgn(aRgn, bRgn, TempRgn);
      SectRgn(cRgn, TempRgn, TempRgn);
      PaintRgn(TempRgn);

      DisposeRgn(aRgn);
      DisposeRgn(bRgn);
      DisposeRgn(cRgn);
      DisposeRgn(TempRgn);
   END;

{Animate the Shape image using AnimatePalette/CLUT resouces.}
   PROCEDURE AnimShape;
      VAR
         count : INTEGER;
         MyCLUT : ARRAY[1..3] OF CTabHandle;
   BEGIN
      FOR count := 1 TO 3 DO
         MyCLUT[count] := GetCTable(count + 300);

      DoDelay(1);
      AnimatePalette(MyWindow[shapeW], MyCLUT[2], 0, 0, 8);
      DoDelay(60);
      AnimatePalette(MyWindow[shapeW], MyCLUT[3], 0, 0, 8);
      DoDelay(60);
      AnimatePalette(MyWindow[shapeW], MyCLUT[1], 0, 0, 8);
      DoDelay(50);

      FOR count := 1 TO 5 DO
         BEGIN
         DoDelay(10);
         AnimatePalette(MyWindow[shapeW], MyCLUT[2], 0, 0,
8);
         DoDelay(10);
         AnimatePalette(MyWindow[shapeW], MyCLUT[3], 0, 0,
8);
         DoDelay(10);
         AnimatePalette(MyWindow[shapeW], MyCLUT[1], 0, 0,
8);
         END;

      DoDelay(60);

      FOR count := 1 TO 5 DO
         BEGIN
         DoDelay(1);
         AnimatePalette(MyWindow[shapeW], MyCLUT[2], 0, 0,
8);
         DoDelay(1);
         AnimatePalette(MyWindow[shapeW], MyCLUT[3], 0, 0,
8);
         DoDelay(1);
         AnimatePalette(MyWindow[shapeW], MyCLUT[1], 0, 0,
8);
         END;

      FOR count := 1 TO 3 DO
         DisposCTable(MyCLUT[count]);
   END;

{********************* Rainbow *********************}

{Demonstrates the Rainbow Effect (Rotating Circle,}
{Moving Bands and Expanding Circle).}
()
{(Create the Rainbow Animation Window.}
   PROCEDURE MakeRainbow;
      VAR
         tempRect : rect;
         tempRGB : RGBColor;
         S : str255;
         tempHSV : HSVColor;
         count : integer;
   BEGIN
      SetRect(tempRect, 50, 160, 590, 400);
      GetIndString(S, StrID, 6);
      MyWindow[rainbowW] := NewCWindow(NIL, tempRect, S, true,
noGrowDocProc, POINTER(-1), true, 0);
      MyPalette[rainbowW] := NewPalette(122, NIL, pmAnimated,
0);

      SetRGB(tempRGB, $FFFF, $FFFF, $FFFF);
      SetEntryColor(MyPalette[rainbowW], 0, tempRGB);
      SetRGB(tempRGB, 0, 0, 0);
      SetEntryColor(MyPalette[rainbowW], 1, tempRGB);
      tempHSV.saturation := $FFFF;
      tempHSV.value := $FFFF;
      FOR count := 1 TO 120 DO
         BEGIN
```

```
            tempHSV.hue := ($0FFFF * count) DIV 120;
            HSV2RGB(tempHSV, tempRGB);
            SetEntryColor(MyPalette[rainbowW], count + 1,
tempRGB);
         END;
      SetPalette(MyWindow[rainbowW], MyPalette[rainbowW],
true);
   END;

{Draws the rays of the Rainbow.}
   PROCEDURE DoRainbowUpdate;
      VAR
         count : INTEGER;
         tempRect, CRect : Rect;
   BEGIN
      SetRect(tempRect, 0, 0, 480, 240);
      PmForeColor(0);
      PaintRect(tempRect);
      SetRect(tempRect, 0, 0, 240, 240);
      SetRect(CRect, 300, 0, 540, 240);
      FOR count := 0 TO 119 DO
         BEGIN
            PmForeColor(count + 2);
            PaintArc(tempRect, count * 3, 3);

            MoveTo(240, count);
            Line(60, 0);
            MoveTo(240, count + 120);
            Line(60, 0);

            PaintOval(CRect);
            InsetRect(CRect, 1, 1);
         END;
   END;

{Rotates all the entries in the CLUT one position.}
   PROCEDURE BumpCTEntry (C : CTabHandle);
      VAR
         tempRGB : RGBcolor;
         MyCT : MyCTabHandle;
         count : INTEGER;
   BEGIN
      MyCT := POINTER(C);
      WITH MyCT^^ DO
         BEGIN
            CopyRGB(ctTable[0].rgb, tempRGB);

            FOR count := 1 TO ctSize DO
               CopyRGB(ctTable[count].rgb, ctTable[count -
1].rgb);

            CopyRGB(tempRGB, ctTable[ctSize].rgb);
         END;
   END;

{Animate the Rainbow using AnimatePalette.  This one}
{creates and manilpulates it's CLUT directly.}
   PROCEDURE AnimRainbow;
      VAR
         count : INTEGER;
         tempRGB : RGBColor;
         tempCT : CTabHandle;
   BEGIN
      tempCT := NewCT(120);
      FOR count := 1 TO 120 DO
         BEGIN
            GetEntryColor(MyPalette[rainbowW], count + 1,
tempRGB);
            SetCTEntry(tempCT, count - 1, tempRGB.red,
tempRGB.green, tempRGB.blue);
         END;

      FOR count := 1 TO 360 DO
         BEGIN
            BumpCTEntry(tempCT);
            DoDelay(1);
            AnimatePalette(MyWindow[rainbowW], tempCT, 0, 2,
120);
         END;
      DisposHandle(Handle(tempCT));
   END;

{******************** Fade ********************}

{Demonstrates the Fade effect}

{Create the Fade Animation Window (uses Palette resource).}
   PROCEDURE MakeFade;
   BEGIN
      MyWindow[fadeW] := GetNewCWindow(FadeID, NIL, POINTER(-
1));
   END;

{Draws Fade window}
   PROCEDURE DoFadeUpdate;
      VAR
         tempRect : Rect;
         count : INTEGER;
   BEGIN
      PmForeColor(0);
      SetRect(tempRect, -32000, -32000, 32000, 32000);
      PaintRect(tempRect);

      FOR count := 1 TO 4 DO
         BEGIN
            PmForeColor(count);
            SetRect(tempRect, ((count - 1) * 100) + 10, 10,
(count * 100) - 10, 90);
            PaintOval(tempRect);
         END;

      FOR count := 5 TO 8 DO
         BEGIN
            PmForeColor(count);
            SetRect(tempRect, ((count - 5) * 100) + 10, 110,
((count - 4) * 100) - 10, 190);
            PaintOval(tempRect);
         END;
   END;

{Animate the Fade.}
   PROCEDURE AnimFade;
      CONST
         FadeStep = 60;
      VAR
         count, E : INTEGER;
         Buf, Inc, Start : ARRAY[0..8] OF RGBColor;
   BEGIN
      SetRGB(Buf[0], -1, -1, -1);
      SetRGB(Buf[1], 0, 0, 0);
      SetRGB(Buf[2], -1, 0, 0);
      SetRGB(Buf[3], 0, -1, 0);
      SetRGB(Buf[4], 0, 0, -1);
      SetRGB(Buf[5], 0, -1, -1);
      SetRGB(Buf[6], -1, 0, -1);
      SetRGB(Buf[7], -1, -1, 0);
      SetRGB(Buf[8], 30000, 30000, 30000);
      FOR E := 0 TO 8 DO
         BEGIN
            CopyRGB(Buf[E], Start[E]);
            UnSignedDiv(Buf[E].Red, FadeStep, Inc[E].Red);
            UnSignedDiv(Buf[E].Green, FadeStep,
Inc[E].Green);
            UnSignedDiv(Buf[E].Blue, FadeStep, Inc[E].Blue);
         END;
```

```
      FOR count := FadeStep - 1 DOWNTO 1 DO
         BEGIN
            FOR E := 0 TO 8 DO
               BEGIN
                  DoDelay(1);
                  UnSignedSub(Buf[E].Red, Inc[E].Red);
                  UnSignedSub(Buf[E].Green, Inc[E].Green);
                  UnSignedSub(Buf[E].Blue, Inc[E].Blue);
                  AnimateEntry(MyWindow[fadeW], E, Buf[E]);
               END;
         END;

      DoDelay(1);
      FOR E := 0 TO 8 DO
         BEGIN
            SetRGB(Buf[E], 0, 0, 0);
            AnimateEntry(MyWindow[fadeW], E, Buf[E]);
         END;

      DoDelay(90);

      FOR count := 1 TO FadeStep - 1 DO
         BEGIN
            FOR E := 0 TO 8 DO
               BEGIN
                  DoDelay(1);
                  UnSignedAdd(Buf[E].Red, Inc[E].Red);
                  UnSignedAdd(Buf[E].Green, Inc[E].Green);
                  UnSignedAdd(Buf[E].Blue, Inc[E].Blue);
                  AnimateEntry(MyWindow[fadeW], E, Buf[E]);
               END;
         END;

      DoDelay(1);
      FOR E := 0 TO 8 DO
         AnimateEntry(MyWindow[fadeW], E, Buf[E]);
   END;

END.
```

```
UNIT PalFunGlobals;

INTERFACE

   USES
      ROM85, ColorQuickDraw, ColorWindowMgr, PaletteMgr,
PickerIntf;

{ Global Constants }
   CONST

      appleM = 301;           {Menu ID Constants}
      fileM = 302;
      editM = 303;
      winM = 304;
      menuCount = 304;

      numWindows = 9;         {Window ID Constants}
      redW = 1;
      greenW = 2;
      blueW = 3;
      curW = 4;
      ballW = 5;
      shapeW = 6;
      rainbowW = 7;
      fadeW = 8;

      StrID = 300;            {Various Resource ID Constants}
      AlertID = 300;
      FadeID = 300;
      ShapeID = 301;

      MaxCT = 2048;  {Max Numbers in CLUT (usually only 256)

      ColorInc = $200; {Amount difference between Color rings}
      ColorStart = $FE00; {Start Color Rings}

(My Version of the CLUT data structure (used for stuffing
values).)
   TYPE
      MyCSpecArray = ARRAY[0..MaxCT] OF ColorSpec;

      MyCTabHandle = ^MyCTabPtr;
      MyCTabPtr = ^MyColorTable;
      MyColorTable = RECORD
            ctSeed : LONGINT;
            ctFlag : INTEGER;
            ctSize : INTEGER;
            ctTable : MyCSpecArray;
         END;

{Standard Variables (Menus, Window Pointer, Window Record)}
{And Palette Handles (1 per Window). }
   VAR
      myMenus : ARRAY[appleM..menuCount] OF MenuHandle;
      MyWindow : ARRAY[1..numWindows] OF WindowPtr;
      MyPalette : ARRAY[1..numWindows] OF PaletteHandle;
      dragRect : Rect;
      doneFlag : BOOLEAN;

IMPLEMENTATION

END.
```




```
* PalFun.r - Resources for PalFun
* by Steve Sheets for MacTutor
*
PalFun.RSRC
????????

Type PASS = STR
  ,0      ;;0 by convention
Palette Animation by Steve Sheets \A9 MacTutor 1988

Type BNDL
  ,128
  PASS  0
  ICN#  1
  0  128
```


© The Definitive MacTutor, Vol. 4

```
  FREF 2
  Ø 128

Type FREF
  ,128
  APPL  Ø ;; local id Ø for icon list

Type SIZE = GNRL
   ,-1
.I
16384  ;; $4000 = bit 14 set
.L
192000 ;; recomended
.L
128000 ;; minimum

Type ALRT
  ,500 (Ø)
40 170 140 470
500
4444

Type DITL
  ,500 (Ø)
2

Button
60 120 80 180
OK

staticText Disabled
20 20 40 280
^Ø

Type ALRT
  ,300 (Ø)
40 170 200 470
300
4444

Type DITL
  ,300 (Ø)
5

Button
120 120 140 180
OK

staticText Disabled
20 73 39 227
PalFun by Steve Sheets

staticText Disabled
40 58 59 242
Designed for MacTutor 3/88

staticText Disabled
60 15 79 285
A demonstration of the Palette Manager

staticText Disabled
80 74 99 226
and Palette Animation.

Type MENU
  ,301 (Ø)
\14
About PalFun...
(-

Type MENU
  ,302 (Ø)
File
Animate/A
Close/W
(-
Quit/Q

Type MENU
  ,303 (Ø)
Edit
Undo
(-
Cut
Copy
Paste
Clear

Type MENU
  ,304 (Ø)
Window
Red
Green
Blue
Current Color
Ball
Shape
Rainbow
Fade

Type STR#
  ,300 (Ø)
6
Red
Green
Blue
Current Colors
Ball
Rainbow

Type WIND
  ,308 (Ø)
Face
120 140 420 520
Visible GoAway
Ø
Ø

Type WIND
  ,300 (Ø)
Fade
100 100 300 500
Visible GoAway
Ø
Ø

Type WIND
  ,301 (Ø)
Shape
140 120 280 370
Visible GoAway
Ø
Ø

Type pltt = GNRL
   , 300
.H
0009 0000 0000 0000 0000 0000 0000 0000
FFFF FFFF FFFF 0004 0000 0000 0000 0000
0000 0000 0000 0004 0000 0000 0000 0000
FFFF 0000 0000 0004 0000 0000 0000 0000
0000 FFFF 0000 0004 0000 0000 0000 0000
0000 0000 FFFF 0004 0000 0000 0000 0000
0000 FFFF FFFF 0004 0000 0000 0000 0000
FFFF 0000 FFFF 0004 0000 0000 0000 0000
FFFF FFFF 0000 0004 0000 0000 0000 0000
7530 7530 7530 0004 0000 0000 0000 0000

Type pltt = GNRL
   , 301
.H
0008 0000 0000 0000 0000 0000 0000 0000
FFFF 0000 0000 0004 0000 0000 0000 0000
0000 0000 FFFF 0004 0000 0000 0000 0000
FFFF 0000 0000 0004 0000 0000 0000 0000
0000 0000 FFFF 0004 0000 0000 0000 0000
FFFF 0000 0000 0004 0000 0000 0000 0000
0000 0000 FFFF 0004 0000 0000 0000 0000
FFFF 0000 0000 0004 0000 0000 0000 0000
0000 0000 FFFF 0004 0000 0000 0000 0000

Type clut = GNRL
   , 301
.H
0000 0000 0000 0007
0000 FFFF 0000 0000
0001 0000 0000 FFFF
0002 FFFF 0000 0000
0003 0000 0000 FFFF
0004 FFFF 0000 0000
0005 0000 0000 FFFF
0006 FFFF 0000 0000
0007 0000 0000 FFFF

Type clut = GNRL
   , 302
.H
0000 0000 0000 0007
0000 FFFF 0000 0000
0001 FFFF 0000 0000
0002 0000 0000 FFFF
0003 0000 0000 FFFF
0004 FFFF 0000 0000
0005 FFFF 0000 0000
0006 0000 0000 FFFF
0007 0000 0000 FFFF

Type clut = GNRL
   , 303
.H
0000 0000 0000 0007
0000 FFFF 0000 0000
0001 FFFF 0000 0000
0002 FFFF 0000 0000
0003 FFFF 0000 0000
0004 0000 0000 FFFF
0005 0000 0000 FFFF
0006 0000 0000 FFFF
0007 0000 0000 FFFF

Type ICN# = GNRL
  ,128 (4)  ;; The Appl. Icon
.H
FFFFFFFF C0000003 BFFFFFFD B000000D
AFFFFFF5 AC000035 ABFFFFD5 AA000055
AA7E0055 AA810055 AA810855 AA811455
AA812255 AA814155 AA7E3E55 AA000055
AA000055 AA007E55 AA008155 AA108155
AA288155 AA448155 AA828155 AA7C7E55
AA000055 ABFFFFD5 AC000035 AFFFFFF5
B000000D BFFFFFFD C0000003 FFFFFFFF
*
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
```

Pascal Procedures

Serial Port Demo

Tom Scheiderich
Fullerton, CA
MacTutor Vol. 4 No. 6

Just try to find out anything about how to talk to the Macs serial ports. I have looked in about 10-15 books and they all seem to forget the these devices exist. You can see how to write to the screen, the disk, the printer, the system clock etc. What happened to the serial ports!!! They aren't so mysterious, as you will see. The only place I found any information about them is in *Inside Macintosh*. Anyone who has read *Inside Macintosh* knows that it is very difficult to follow as a tutorial, even by professional programmers. In many cases, a good example program, along with some discussion of the subject, will better help most programmers grasp the new techniques. "A Picture is worth a Thousand Words"!!! I learned about serial ports from a very basic terminal emulator. I just played with it and talked to other people about it. I will try to impart what I have learned to those interested. The sample program is derived from a Data Analyser program I wrote which can be used to examine data going back and forth between two serial devices. For example, from between a printer and a computer or a modem and a computer. Our sample program displays one window for each serial port, and sends data typed at the keyboard out one port and in the other, where it is displayed in the port's window. This completely demonstrates both reading and writing to the Macintosh serial ports.

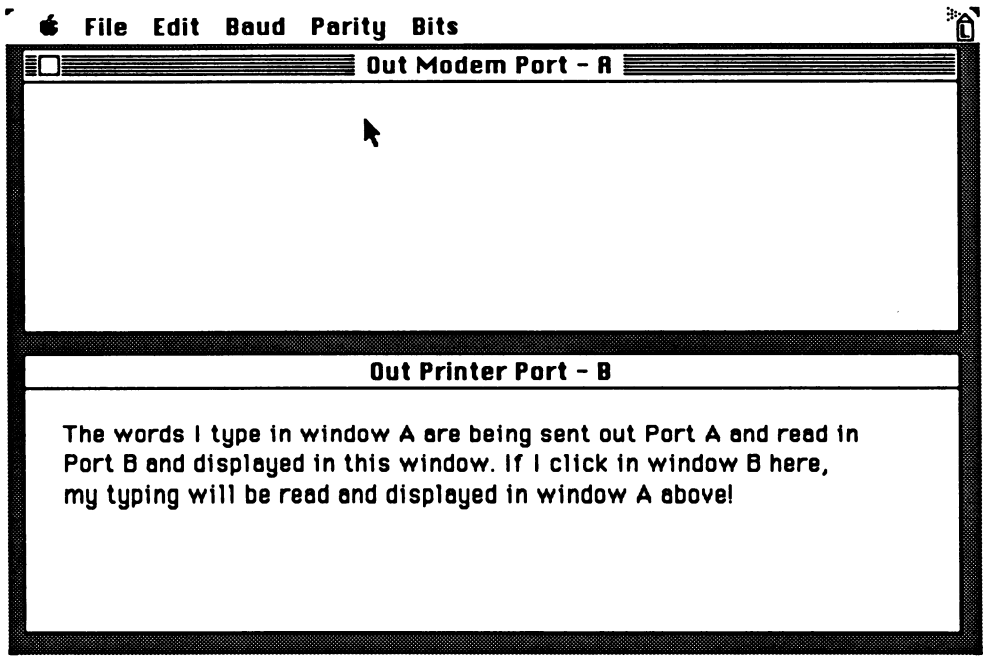


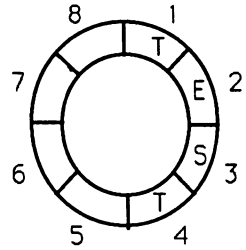
Fig. 1 Output of our sample Serial Port Program

A serial port is one of the computer's means to talk to the outside world in which the data is sent one bit at a time to some device such as a modem or printer. The Mac has 2 such ports at its disposal. Each port has two default buffers of 64 characters, an input and an output buffer. It seems that each buffer is set up in a circular fashion.

General Circular Buffer Procedure

A circular buffer works logically as shown in the graphic. Logically there is no end to the buffer. It just keeps going around in a circle.

What actually happens in this type of buffer is that as a byte is received, it is put in the next position in the buffer. If the end of the buffer is reached, the byte is put at the start of the buffer overwriting whatever byte was there before. The users extracts these bytes from this buffer and put them into a work area (a character, byte or array). As each byte is received, a check must be made to see if the number of bytes in the buffer match the maximum number of bytes allowed in the buffer. Since we don't want to over- write these bytes, the pro- gram must then tell the device (modem, for example) not to send any more data until later. The device will then "go to sleep" until requested to continue. This is called **hand- shaking.** There is no shift- ing of bytes as the data is read from and writ- ten to the buffer. The bytes will stay in the buffer as is, until over-written by new data.



In the following example, we will assume that data is coming from a modem and is sending the word "TESTING". We have already received "TEST" and we are just going to read our first byte from the buffer. There are four control variables needed to handle the buffer.

1. Buffer Size - This is the total number of bytes available in our buffer.

© The Definitive MacTutor, Vol. 4

2. Get Byte - The next byte position in the buffer to read.
3. Put Byte - The next byte position in the buffer available to put a byte.
4. Byte Count - The number of bytes available in the buffer to read. This will be (Put Byte - Get Byte). If < 0 then add Buffer Size.

CIRCULAR BUFFER EXAMPLE

Assume the buffer already contains 4 bytes - 'TEST' and the buffer is 64 bytes long

Serial Input Buffer

T	E	S	T				
1	2	3	4	5	6	7	8

BUFFER SIZE - 64
BYTE COUNT - 4
GET BYTE - 1
PUT BYTE - 5

At this point there are 4 bytes in the Buffer. The next byte we are going to read is a "T" which is in byte #1. The next byte coming from the modem will go into byte #5.

After reading a character

T	E	S	T				
1	2	3	4	5	6	7	8

BUFFER SIZE - 64
BYTE COUNT - 3
GET BYTE - 2
PUT BYTE - 5

PROGRAM I/O BUFFER | T | | |

After the byte is taken from the circular buffer and put into a buffer specified by the program, the Byte Count will be set to 3, which says there are now three bytes in the buffer to be read. The next byte to be read is in the 2nd byte position. This will be the letter "E". You will notice that nothing has happened to the "T" in position 1.

After writing a character

T	E	S	T	I			
1	2	3	4	5	6	7	8

BUFFER SIZE - 64
BYTE COUNT - 4
GET BYTE - 2
PUT BYTE - 6

Next character read from modem | I |

After the byte is put into the circular buffer, the Byte Count is then set to 4, which says there are now four bytes in the buffer to be read. The next byte read from the modem will be put into the 6th byte position.

After writing a character, at the end of the buffer

R	E	S	T	I		F	O
1	2	3	4	5		63	64

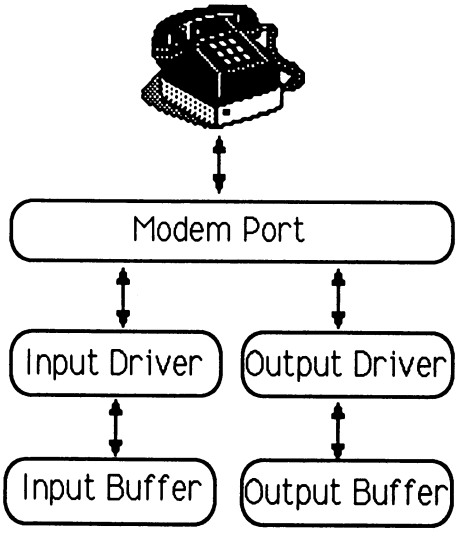
BUFFER SIZE - 64
BYTE COUNT - 64
GET BYTE - 2
PUT BYTE - 2

Next character read from modem | R |

Here we are also putting a byte into the buffer but have run out of room. The next position would have been 65. Since this is greater than the Buffer Size, we now wrap around and put the byte in byte position 1, over-writing the "T". Also, you will notice that the Byte Count equals the Buffer size. We cannot accept any more bytes until some bytes are taken out of the buffer, so we must perform some sort of handshaking to stop the modem from sending any more data. In the real world, this would happen before the buffer was full. The data could be coming so fast that by the time the modem has found out it is supposed to stop, it may have sent 20 more bytes.

Macintosh Serial Ports

Now we will see how this all applies to the Mac, which of course is our only reason for living.



The Mac has two serial ports to choose from. Each of these ports has an input buffer and an output buffer. These are "our" circular buffers. Each buffer has a default size of 64 bytes. This can be changed to whatever size you want by a call to **SerSetBuf**. You don't have to worry about the control variables from our previous discussion of the circular buffers as these are handled internally by the Serial Drivers. The control variables were just to demonstrate how a circular buffer is handled. You may, however, want to know how many bytes are in the buffer at any one time. This is accomplished by a call to **SerGetBuf**. If the count returned is not equal to 0, then a call to **FSRead** is executed. **FSRead** will take the number of bytes requested out of the buffer, puts them into a buffer specified by the program and updates the buffers control words. The program can then do what it wishes with the data. For example, if running a terminal emulator, the bytes received might be text and can now be displayed to the screen. If receiving a file, the bytes might now be written to the disk.

The sample program was written in Lightspeed Pascal. You should be able to see that dealing with the serial ports is very simple and not a mystery at all. The whole serial flow can be broken down to the following steps:

1. Open and set up the Serial Port.
2. Get any bytes that might be in the input buffer and deal with the data.
3. Check if any keys have been pressed and write out the serial port
4. Go back to # 2 and repeat.

The actual flow is summarized here in the flowchart shown at the end of the article.

In the example program, characters entered by the keyboard are written out one serial port and read into the other serial port and displayed on one of the two screens.

Before we start looking at the program some important Serial port variables need to be understood:

1. inBuffPtr - this is the 2k circular buffer that we are going to replace the default 64k buffer with. We set up the size and pointer to it (NewPtr) and then pass the pointer to the driver routine by way of **SerSetBuf** . We don't deal with it anymore from this point on. All of our dealings will be with inRefNum and outRefNum
2. filterBuffPtr - the data returned from **FSRead** is put into this buffer. Since we will only be reading one character at a time, we only set up the buffer as one byte long. If we were going to read more than one byte from the buffer, you must set it up to the maximum number of bytes you might read. You may want to set the size the same as inBuffPtr . Unlike the circular buffers, the data being transfered will be put in the buffer starting from the first byte and continuing until all the bytes requested are transferred.
3. inRefNum - this is the input channel (reference) number. This number will be either a -6 (modem) or -8 (printer). All reading from the serial port will be done by referencing this number. It gets set up in **RAMSDopen**.
4. outRefNum - this is the output channel (reference) number. This number will be either a -7 (modem) or -9 (printer). All writing to the serial port will be done by referencing this number. It gets set up in **RAMSDopen**.

Each of these variables is set up as either "A" variables for the modem ports or "B" variables for the printer ports. For example, there is an inRefNumA for the modem and an inRefNumB for the printer. These are needed in our routine since we will be reading and writing to both of the ports.

To change the default settings (9600 baud, 8 data bits, 2 stop bits and no parity), set **SerConfig**, which is an integer that has its bits set for the appropriate parameters and passed along with the reference number to **SerReset**.

Handshaking is needed to prevent data overruns in the circular buffers. If no handshaking or the wrong handshaking is defined, you could get this overrun. Many of you have seen this with your printers when you are struggling to get the correct parity and stop bits set up. You will see a lot of garbage on your paper until you get it correct. This is caused many times because some data is getting lost when the printer has told your computer to stop sending, but because of incorrect settings, the computer kept sending data anyway and some of it was lost.

You will need to set up a record of control bytes to tell the Mac what type of handshaking you want it to do and call **SerHShake** while passing this record. The record contains:

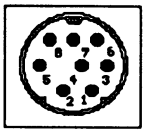
1. fXOn - XOn/XOff output buffer flow control flag. If this is nonzero, then XOn/XOff flow control is enabled for the output buffer.
2. fCTS - If nonzero, then hardware flow control is enabled. The handshaking will then be controlled by lines 4 and 5 of the serial port.
3. xOn - What XOn character to use (usually a control-Q).
4. xOff - What XOff character to use (usually a control-S).
5. errs - Tells the driver which type of errors cause input request to be aborted (parity, hardware overruns, or framing errors).
6. evts - This byte tells whether changes in CTS or Break status will cause the driver to post device driver events. This option is discouraged because interrupts are disabled for a long time while events are being posted
7. fInX - XOn/XOff input buffer flow control flag. If this is nonzero then XOn/XOff flow control is enabled for the input buffer.

The type of handshaking we use here doesn't matter because everytime we send a character we also read it, so there is no way to get an overrun. But I have it set up as XOn/XOff for the input buffer only, using a control-S and a control-Q as our XOn/XOff characters for demonstration purposes.

Before starting the program you will need to a special cable. They are very easy to build. Since we are dealing with XOn/

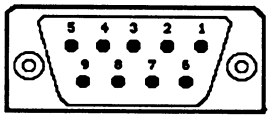
XOff, we need only 3 wires; the transmit, receive and ground wires. We are building a null modem cable which is just a cable which reverses the transmit and receive lines. When we are transmitting out the modem port, we will be receiving through the printer port and vice versa.

MINI-8



1 Output Handshake
2 Input handshake/external clock
3 Transmit data -
4 Ground
5 Receive data -
6 Transmit data +
7 (not connected)
8 Receive data +

DB-9



1 Ground
2 +5 volts
3 Ground
4 Transmit data +
5 Transmit data -
6 +12 volts
7 Handshake/external clock
8 Receive data +
9 Receive data -

Pin Assignments on the Serial Ports

Following are the cable setups needed for the included program. To use the DB-9's, if you are using a Mac Plus, Mac SE or a Mac II, you will need to purchase two - DB-9 to mini-8 conversion cables. The DB-9 connectors are easier to deal with than the mini-8 connecters and both DB-9 connectors are male.

To wire the cable, take the two male DB-9 connectors and wire them as shown in the figure below. Wire pin 3 to pin 3, ground. Wire pin 5 on one to pin 9 on the other for transmit. And finally, wire pin 9 on one to pin 5 on the other for receive. Then plug each male DB-9 connector into an Apple mini-8 to DB-9 cable and plug the two mini-8 connectors into the modem and printer ports on the back of the Mac. The Apple cables are available at Apple dealers to convert the mini-8 serial ports to the old style DB-9 connector devices used by the Mac 128 and Mac 512 computers. If you are a wiz, you can try to make a mini-8 cable instead, but we don't recommend it.

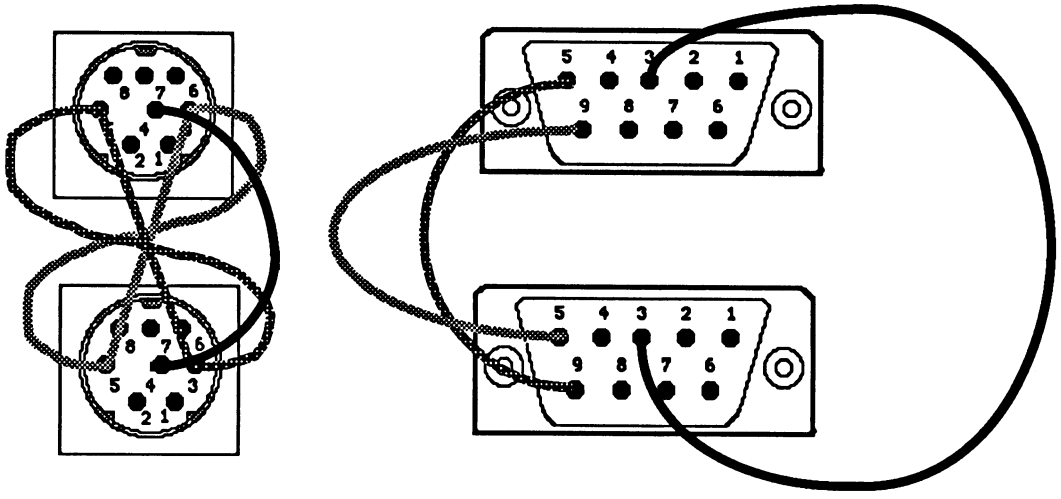
The first thing that is done in our program is to open the serial ports which is done in the Init_Serial routine. We are using the RAM serial driver. *Note that AppleTalk must be turned off at the chooser in order for this to work.* All buffers and pointers are set up here. Next we will set up the two windows shown in figure 1.

The window selected is the port the data will be transferred out of and the other window is the port that will be doing the receiving. The non-selected window will be the one we display the text on.

Now we will just go into our normal "mainloop" routine. The first thing we will do is check for any bytes in the serial buffer (Get_comm_input). **SerGetBuf** is called to see if there are any bytes in our serial buffer. If there are, we will call **FSRead** (from GetChar) to get a character out of the buffer. Before writing to the window we need to call SetPort to select the correct window. If **PortA** is true, we are writing out the modem port and are displaying on window B, so we would then call **SetPort**(windowB) to select the second window and draw the character onto the screen (Put_Char).

Last of all is to see if a key was pressed. If so, we need to write the character out the output port (Handle_keys). A call would be made to FSWrite to transfer the character.

The only other routine of interest would be RSerBuf which is called to reset the buffers and pointers and to reverse the input and output ports. This is done whenever a new window is selected or at the start of the program. Clicking in the content region of the unselected window, then reverses which window and port is doing the output and input of our characters typed on the keyboard.

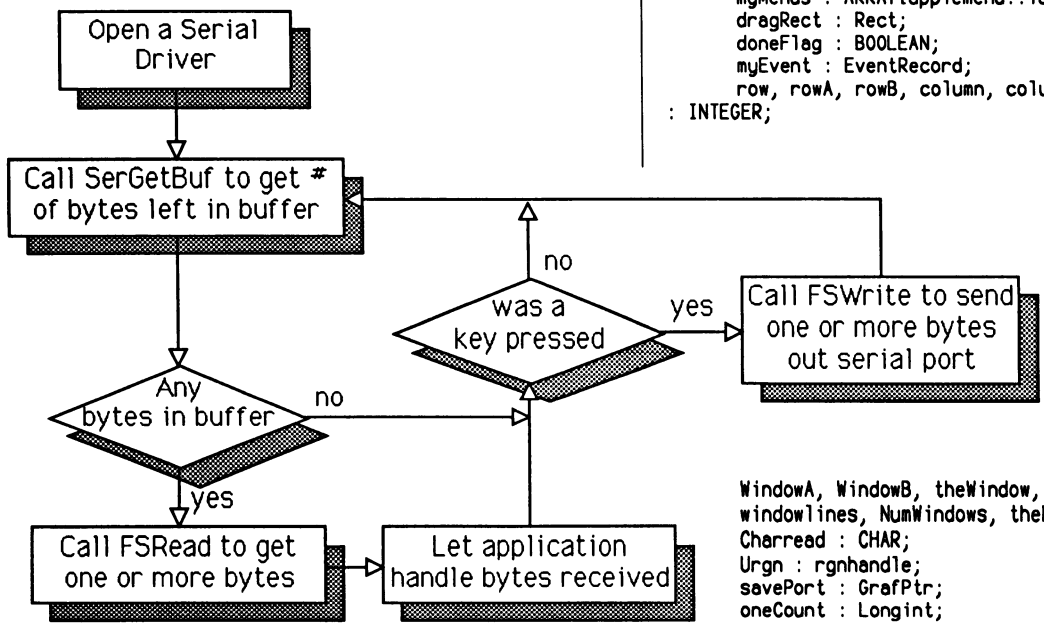


Wiring our null modem cable for port to port communications

One last thing. We are using the RAM Serial Driver for our program, but we could just as easily have used the ROM Serial Driver. The main problem is that the ROM Driver doesn't support XOn/XOff, so if this is necessary use the RAM Driver. If the ROM driver is needed then all that need be done is:

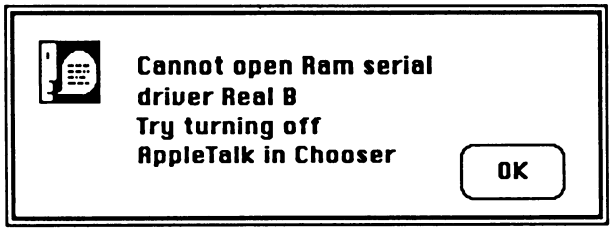
```
Replace
    errI:=RAMSDOpen(sPortA)
With
    errI := OpenDriver('.Ain', inRefNum);
            errI := OpenDriver('.Aout', outRefNum);
```

You need to explicitly open the input and the output Drivers.



Flowchart for serial port operation

```
UNIT Globals;

INTERFACE

  CONST
    rsrc = 'SERL';
    aAbout = 1;
    lastMenu = 408; { number of menus }
    appleMenu = 403; { menu ID for desk accessory menu }
    fileMenu = 404; { menu ID for File menu }
    editMenu = 405; { menu ID for Edit menu }
    baudMenu = 406;
    MessageDialog = 256;
    AboutDialog = 257;
    b300 = 1;
    b1200 = 2;
    b2400 = 3;
    b4800 = 4;
    b9600 = 5;
    b19200 = 6;
    parityMenu = 407;
    noP = 1;
    evenP = 2;
    oddP = 3;
    bitsMenu = 408;
    bData7 = 1;
    bData8 = 2;
    bStop1 = 4;
    bStop15 = 5;
    bStop2 = 6;

{- - - - -  serial port - - - - - - - - }
    inBuffLength = 2048;

    XONCHAR = 17;
    XOFFCHAR = 19;
    CrRtn = 13;
    LineFeed = 10;


  VAR
    myMenus : ARRAY[applemenu..lastMenu] OF MenuHandle;
    dragRect : Rect;
    doneFlag : BOOLEAN;
    myEvent : EventRecord;
    row, rowA, rowB, column, columnA, columnB, code, refNum
: INTEGER;

    WindowA, WindowB, theWindow, whichWindow : WindowPtr;
    windowlines, NumWindows, theMenu, theItem : INTEGER;
    Charread : CHAR;
    Urgn : rgnhandle;
    savePort : GrafPtr;
    oneCount : Longint;

{—— serial port variables ————}
    inBuffPtr, filterBuffPtr, outBuffPtr : Ptr;
    inBuffPtrA, filterBuffPtrA, outBuffPtrA : Ptr;
    inBuffPtrB, filterBuffPtrB, outBuffPtrB : Ptr;
    inRefNumA, outRefNumA : INTEGER;
    inRefNumB, outRefNumB : INTEGER;
    inRefNum, outRefNum, err : INTEGER;
    serConfig,    (sum of following vars)
      baud, parity, dataBits, stopBits : INTEGER;
    portA : Boolean;

IMPLEMENTATION

END.
```

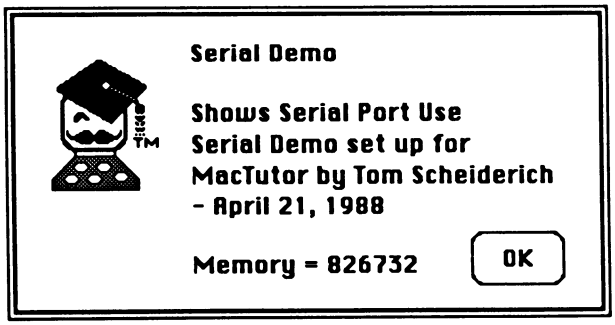


Typical Error produces this error message

© The Definitive MacTutor, Vol. 4

```
UNIT initprogram;

INTERFACE

   USES
      Globals, Misc;

   PROCEDURE initall;

IMPLEMENTATION

   PROCEDURE SetUpMenus;

      VAR
         i : INTEGER;

   BEGIN
      InitMenus; { initialize Menu Manager }
      myMenus[applemenu] := GetMenu(appleMenu);
      AddResMenu(myMenus[applemenu], 'DRVR');
      { desk accessories }
      myMenus[filemenu] := GetMenu(fileMenu);
      myMenus[editMenu] := GetMenu(editMenu);
      myMenus[baudMenu] := GetMenu(baudMenu);
      myMenus[parityMenu] := GetMenu(parityMenu);
      myMenus[bitsMenu] := GetMenu(bitsMenu);

      FOR i := applemenu TO lastMenu DO
         InsertMenu(myMenus[i], 0);
      DrawMenuBar;
   END; { of SetUpMenus }

{—— set up serial port parameters ——}

   PROCEDURE Init_serial;

      VAR
         errI : INTEGER;
         Hand : SerShk;

   BEGIN

   { Pointers to buffers need to be set up - we are setting up
for 2k and overriding 64 byte default. Note very simple error
checking using a vanilla dialog box routine doMessage. }

      inBuffPtrA := NewPtr(inBuffLength);
      filterBuffPtrA := NewPtr(1);
      inBuffPtrB := NewPtr(inBuffLength);
      filterBuffPtrB := NewPtr(1);

      errI := RAMSDOpen(sPortA);
      { open modem port - sPortA is already defined }
      IF errI <> noErr THEN
         doMessage('Cannot open Ram serial driver Real A',
'Try turning off Appletalk in Chooser', '', '');

      errI := RAMSDOpen(sPortB);
      { open printer port - sPortB is already defined }
      IF errI <> noErr THEN
         doMessage('Cannot open Ram serial driver Real B',
'Try turning off AppleTalk in Chooser', '', '');

   inRefNumA := AinRefNum; {driver reference number = -6}
   outRefNumA := AoutRefNum; {driver reference number = -7}

   inRefNumB := BinRefNum; {driver reference number = -8}
   outRefNumB := BoutRefNum; {driver reference number = -9}

   {set baud rate, parity, etc. for both in and out}

      baud := 94;            {1200}
      parity := 0;           {none}
      dataBits := 3072;      {8}
      stopBits := 16384;     {1}
      serConfig := baud + parity + dataBits + stopBits;

{ Note we are lazy now, no error checking! You should check }
      errI := SerReset(inRefNumA, serConfig);
{ set modem input }
      errI := SerReset(outRefNumA, serConfig);
{ set modem output }
      errI := SerReset(inRefNumB, serConfig);
{ set printer input }
      errI := SerReset(outRefNumB, serConfig);
{ set printer output }

      { set handshaking of ports for xon/xoff }

      WITH Hand DO
         BEGIN
            FXON := 0;
            FCTS := 0;
            XON := CHAR(XONCHAR);
            XOFF := CHAR(XOFFCHAR);
            ERRS := 0;
            EVTS := 0;
            FINX := 1;
         END;

      ErrI := SerHShake(inRefNumA, Hand);
      ErrI := SerHShake(inRefNumB, Hand);

      {set buffer}

      errI := SerSetBuf(inRefNumA, inBuffPtrA, inBuffLength);
      errI := SerSetBuf(inRefNumB, inBuffPtrB, inBuffLength);

      { initially set I/O ports to the modem ports }

      inRefNum := inRefNumA;
      outRefNum := outRefNumB;
   END;  {Init_serial}

   PROCEDURE initall;
      VAR
         err : integer;
   BEGIN
      MoreMasters; {create 64 master pointers - for future}
      MoreMasters;
      MaxApplZone;
      InitGraf(@thePort);
      InitFonts;
      FlushEvents(everyEvent, 0);
      InitWindows;
      SetUpMenus;
      init_serial;
      TEInit;
      InitDialogs(NIL);
      HideAll;
      Urgn := newRgn;
      SetCursor(arrow);
      SetRect(dragRect, 4, 24, 508, 338);
      doneFlag := FALSE;
      portA := true;
      row := 1;
      oneCount := 1;
      windowlines := 9;
      column := 1;
      numWindows := 0;
   END;
END.
```

```
UNIT menus;

INTERFACE

   USES
      Globals, GPLib, Misc;

   PROCEDURE DoCommand (mResult : LongInt);

IMPLEMENTATION

   PROCEDURE doAbout;
      VAR
         IDStrHandle : StringHandle;
         dialogP : DialogPtr;
         item : integer;
         Str1, Str2, Str3 : str255;
         myHeapSpace : LongInt;
         FreeSpace : Size;
   BEGIN
      IDStrHandle := StringHandle(GetResource(rsrc, 0));
      IF IDStrHandle = NIL THEN
         BEGIN
            doMessage('Get About box crash!', '', '', '');
            ExitToShell;
         END;
      MoveHHi(Handle(IDStrHandle));
      HLock(Handle(IDStrHandle));
      FreeSpace := FreeMem;
      myHeapSpace := MaxMem(FreeSpace);
      NumToString(myHeapSpace, Str2);
      Str2 := concat('Memory = ', Str2);
      Str3 := '';
      Str1 := '';
      ParamText(IDStrHandle^^, Str1, Str2, Str3);
      dialogP := GetNewDialog(AboutDialog, NIL, pointer(-1));
      IF dialogP = NIL THEN
         BEGIN
         doMessage('Dialog crash!', 'We are dead...', '',
'');
         ExitToShell;
         END;
      initCursor;
      ModalDialog(NIL, item);
      DisposDialog(dialogP);
      HUnlock(Handle(IDStrHandle));
   END;
```



About box code above puts up this dialog

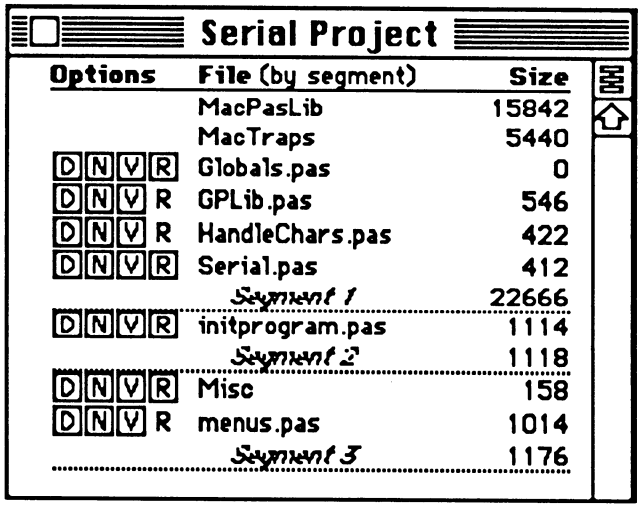
```
   PROCEDURE Do_comm_commands (whichMenu, whichItem :
INTEGER);

      VAR
         errC : INTEGER;
         resetmodem : boolean;

   BEGIN
      resetmodem := false;
      CASE whichMenu OF
         baudMenu :
            BEGIN
               Place_checkmarks(baudMenu, b300, b19200,
whichItem);
               resetmodem := true;
               CASE whichItem OF
                  b300 :
                     baud := 380;
                  b1200 :
                     baud := 94;
                  b2400 :
                     baud := 46;
                  b4800 :
                     baud := 22;
                  b9600 :
                     baud := 10;
                  b19200 :
                     baud := 4;
               END;  {CASE whichItem}
            END;  {baudMenu}

         parityMenu :
            BEGIN
               Place_checkmarks(parityMenu, noP, oddP,
whichItem);
               resetmodem := true;
               CASE whichItem OF
                  noP :
                     parity := 8192;
                  evenP :
                     parity := 12288;
                  oddP :
                     parity := 4096;
               END;  {CASE whichItem}
            END;  {parityMenu}

         bitsMenu :
            BEGIN
               resetmodem := true;
               IF whichItem < 3 THEN
                  Place_checkmarks(bitsMenu, bData7, bData8,
whichItem)
               ELSE
                  Place_checkmarks(bitsMenu, bStop1, bStop2,
whichItem);
               CASE whichItem OF
                  bData7 :
                     dataBits := 1024;
                  bData8 :
                     dataBits := 3072;
                  bStop1 :
                     stopBits := 16384;
                  bStop15 :
                     stopBits := -32768;
                  bStop2 :
                     stopBits := -16384;
               END;  {CASE whichItem}
            END;  {bitsMenu}

      END;  {CASE whichMenu}

      IF resetmodem THEN
         BEGIN
            serConfig := baud + parity + dataBits + stopBits;
            errC := SerReset(inRefNumA, serConfig);
            errC := SerReset(outRefNumA, serConfig);
            errC := SerReset(inRefNumB, serConfig);
            errC := SerReset(outRefNumB, serConfig);
         END;
```

```
    END;  {Do_comm_commands}

    PROCEDURE DoCommand;      {(mResult : LongInt)}
       VAR
          name, command : STR255;

    BEGIN
       theMenu := HiWord(mResult);
       theItem := LoWord(mResult);
       CASE theMenu OF
          appleMenu :
             IF theItem = aAbout THEN
                doAbout
             ELSE
                BEGIN
                   GetItem(myMenus[applemenu], theItem,
name);
                   refNum := OpenDeskAcc(name);
                END;
          fileMenu :
             CASE theItem OF
                7 :
                   doneFlag := TRUE; { Quit }
             END;      { theItem case }

          OTHERWISE
             Do_comm_commands(theMenu, theItem);
       END; { of menu case }

       HiliteMenu(0);
    END; { of DoCommand }
END.


UNIT GPLib;

INTERFACE

    USES
       Globals;

    PROCEDURE RSerBuf (outref : integer);
    PROCEDURE GoToXY (x, y : integer);
    PROCEDURE Place_checkmarks (menuId, firstItem, lastItem,
       chosenItem : Integer);
    PROCEDURE NewWindows;
    PROCEDURE SelectNewWindow;

IMPLEMENTATION

          { reset the I/O variables and buffers }

    PROCEDURE RSerBuf;    {(outref : integer);}
    BEGIN
       IF outrefnum <> outref THEN
       { reset port I/O variables and buffers }
          BEGIN
             IF outrefNumA = outref THEN
       (set to the 'A' buffers and variables }
                BEGIN
                   outrefNum := outrefNumA;
                   inrefNum := inrefNumB;
                   inBuffPtr := inBuffPtrB;
                   filterBuffPtr := filterBuffPtrB;
                   RowA := Row;
                   row := RowB;
                   ColumnA := Column;
                   Column := ColumnB;
                   SetPort(WindowB);
                   theWindow := WindowB;
                END
             ELSE
                BEGIN
               (set to the 'B' buffers and variables }
                      outrefNum := outrefNumB;
                      inrefNum := inrefNumA;
                      inBuffPtr := inBuffPtrA;
                      filterBuffPtr := filterBuffPtrA;
                      RowB := Row;
                      row := RowA;
                      ColumnB := Column;
                      Column := ColumnA;
                      SetPort(WindowA);
                      theWindow := WindowA;
                   END;
          END;
    END;

    { set the position for the next character to be displayed }

    PROCEDURE GoToXY;
       VAR
          I, J : integer;
    BEGIN
       I := x * 6;
       J := y * 12 - 2;
       moveto(I, J);
    END;

    PROCEDURE Place_checkmarks;    {(menuId, firstItem,
lastItem, chosenItem : Integer)}
       VAR
          index : Integer;

    BEGIN
       FOR index := firstItem TO lastItem DO
          checkItem(myMenus[menuID], index, False);
       CheckItem(myMenus[menuID], chosenItem, True);
    END;  {Place_checkmarks}


    { open a new window and set it's characteristics }

    PROCEDURE NewWindows;

    BEGIN
       numWindows := numWindows + 1;
       IF numWindows = 1 THEN
          { modem port A }
          theWindow := GetNewWindow(254, NIL, Pointer(-1));
       IF numWindows = 2 THEN
          { printer port B }
          theWindow := GetNewWindow(255, NIL, Pointer(-1));
       setPort(theWindow);
       TextFont(monaco);
       TextSize(9);
       ShowWindow(theWindow);
       Row := 1;
       Column := 1;
    END;

    { select the window selected by the user }

    PROCEDURE SelectNewWindow;

    BEGIN
       SelectWindow(whichWindow);
       theWindow := whichWindow;

       IF theWindow = windowA THEN
          BEGIN
             portA := true;
             RSerBuf(outrefnumA);
             { reset the I/O buffers and variables }
          END
```

```
      ELSE
         BEGIN
            portA := false;
            RSerBuf(outrefnumB);
            { reset the I/O buffers and variables }
         END;
   END;
END.


UNIT HandleChars;

INTERFACE

   USES
      Globals, GPLib;

   PROCEDURE DisplayChar (charread : char);
   PROCEDURE Put_Char (tempchar : integer;
                       CharRead : Char);
   PROCEDURE Handle_keys (keyPressed : Char);
   PROCEDURE Get_comm_input;

IMPLEMENTATION

   PROCEDURE DisplayChar; {(charread : char)}
   BEGIN
      gotoxy(column, row);
      drawchar(charread);
      column := column + 1;
   END;

   PROCEDURE Put_Char;    { (tempchar : integer;CharRead :
Char)}

   BEGIN
      CASE tempchar OF
         CrRtn, LineFeed :
            BEGIN
            IF row <= windowlines THEN
               row := row + 1
            ELSE
               BEGIN
               ScrollRect(thePort^.portRect, 0, -12, Urgn);
               END;
            Column := 1;
            gotoxy(column, row);
            END;
         OTHERWISE
            DisplayChar(CharRead);
      END;         {tempchar}
   END;

{— handle all keyDown events, including command-keys —}
   PROCEDURE Handle_keys; {(keyPressed : Char)}

      VAR
         oneChar : LongInt;
         charAddr : Ptr;

   BEGIN
      onechar := 1;
      charAddr := POINTER(1 + ORD4(@keyPressed));
      {char in low-order byte of word}

      err := FSWrite(outRefNum, oneChar, charAddr);
      {send out serial port}
   END;     {Handle_keys}


{ get the next character from the input buffer currently
selected }

   PROCEDURE GetChar (VAR xchar : char);
      VAR
         errorCode : Integer;

   BEGIN
      errorCode := FSRead(inRefNum, oneCount, filterBuffPtr);
      Xchar := Chr(filterBuffPtr^);
      { get the character pointed to by filterBuffPtr }
      BitClr(@XChar, 8);  {clear the high bit for ascii data,
not used for binary data}
   END;

{——get characters from serial port, and filter ——}

   PROCEDURE Get_comm_input;

      VAR
         errorCode, tempchar : INTEGER;
         inCount : LongInt;

   BEGIN
      errorCode := SerGetBuf(inRefNum, inCount);
      {get # of char in input buffer}

{ — use the code below to very SLOWLY strip line feed
characters — }
      IF inCount > 0 THEN
         BEGIN
            GetChar(Charread);
            IF ord(charRead) <> 0 THEN
               BEGIN
                  tempchar := BitAnd(ord(charRead), 127);
               { clear the high bit }
                  IF (portA) THEN
                     setport(windowB)
   { we are writing out port A, we are reading port B }
                  ELSE
                     setport(windowA);
   { we are writing out port B, we are reading port A }
                  Put_Char(tempchar, charRead)
               END;  {if charRead}
         END;  {if incount}
   END;     {Get_Comm_input}
END.

UNIT Misc;

INTERFACE

   USES
      Globals;

   PROCEDURE doMessage (message0 : str255;
                        message1 : str255;
                        message2 : str255;
                        message3 : str255);

IMPLEMENTATION

   PROCEDURE doMessage; {(message0 : str255)
                         (message1 : str255)
                         (message2 : str255)
                         (message3 : str255)}
      VAR
         dialogP : DialogPtr;
         item : integer;
   BEGIN
   ParamText(message0, message1, message2, message3);
   dialogP := GetNewDialog(MessageDialog, NIL, pointer(-1));
      IF dialogP = NIL THEN
         BEGIN
            SysBeep(5);
            ExitToShell;
```


© The Definitive MacTutor, Vol. 4

```
      END;
    initCursor; (change to arrow)
    ModalDialog(NIL, item);
    DisposDialog(dialogP);
  END;
END.
```



Units are linked as shown above

```
PROGRAM Serial_Demo;

{  Pascal source:  Serial.Pas}
{  Resources:      Serial.R}

{$R-}
{$I-}
  USES
    Globals, GPLib, HandleChars, initprogram, Menus, Misc;

  PROCEDURE mainloop;
  BEGIN
    SystemTask;
    Get_comm_input;
    IF GetNextEvent(everyEvent, myEvent) THEN
      CASE myEvent.what OF
        mouseDown :
          BEGIN
            code := FindWindow(myEvent.where,
whichWindow);
            CASE code OF
            inMenuBar :
              DoCommand(MenuSelect(myEvent.where));
            inSysWindow :
              SystemClick(myEvent, whichWindow);
            inDrag :
              IF whichWindow <> FrontWindow THEN
                SelectNewWindow
              ELSE
                DragWindow(whichWindow,
myEvent.where, dragRect);

            inContent :
              IF whichWindow <> FrontWindow THEN
                SelectNewWindow;
            END;{ of code case }
          END; { of mouseDown }
        keyDown, autoKey :
          Handle_keys(chr(LoWord(myEvent.message)));

        activateEvt :
          BEGIN
            whichWindow := WindowPtr(myEvent.message);
            IF ODD(myEvent.modifiers) THEN
              SetPort(whichWindow);
          END; { of activateEvt }

        updateEvt :
          BEGIN
            GetPort(savePort);
            whichWindow := WindowPtr(myEvent.message);
            SetPort(whichWindow);
            BeginUpdate(whichWindow);
            EndUpdate(whichWindow);
            SetPort(savePort);
          END { of updateEvt }
      END { of event case }
  END;

BEGIN { main program }
  initall;
  UnloadSeg(@initall);
  newWindows;  { open the Printer window - B - first }
  rowB := row;
  columnB := column;
  WindowB := theWindow;
  newWindows;  { open the modem window -A}
  rowA := row;
  columnA := column;
  WindowA := theWindow;
  RSerBuf(outrefnumA);
  REPEAT
    mainloop;
  UNTIL doneFlag
END.
```

```
*  Serial.R
*

Serial/Rsrc.rsrc
????????

Type SERL = STR
,0
Serial Demo set up for MacTutor by Tom Scheiderich  - April
21, 1988

Type FREF
,128
APPL 0

Type BNDL
,128
SERL 0
ICN#
0 128
FREF
0 128

Type MENU
  ,403
\14    ;;apple menu
   About Serial…
   (-

  ,404
File
  (Open
  (New Window
```

```
  (-
  (Save
  (Save As...
  (-
  Quit


   ,405
Edit
   (Undo/Z
   (-
   (Cut/X
   (Copy/C
   (Paste/V

   ,406
Baud
   300
   1200!\12
   2400
   4800
   9600
   19200

   ,407
Parity
   None!\12
   Even
   Odd

   ,408
Bits
   7 Data
   8 Data!\12
   (-
   1    Stop!\12
   1.5 Stop
   2    Stop


Type WIND
  ,255
Out Modem Port - A
40 10 170 500
inVisible NoGoAway
0
0

Type WIND
  ,254
Out Printer Port - B
200 10 330 500
inVisible NoGoAway
0
0

* —— Multifinder events ——

* bit 15 = switcher save screen
* bit 14 = accept suspend resume events
* bit 13 = switcher enable option switch
* bit 12 = can do background on null
events
* bit 11 = multifinder aware
*         (activates & deactivates
topmost
*          window at resume, suspend
events)

Type SIZE = GNRL
   ,-1
.H
4800      ;; $4800 = bits 14,11 set
.L
128000 ;; (for 150K recomended)
.L
80000  ;; (for 80K minimum)
.I

* Program Messages Dialog box...
type DLOG
   ,256
Program Messages
100 100 200 400
Visible NoGoAway
1
0
256

type DITL
   ,256
3
BtnItem Enabled
65 230 95 285
OK

StatText Disabled
15 60 85 222
^0\0D^1\0D^2\0D^3

IconItem Disabled
10 10 42 42
1

* —— Dialogs ——
* About Box dialog...
type DLOG
   ,257
About Serial…
100 100 250 400
visible NoGoAway
1
0
257

type DITL
   ,257
3
BtnItem Enabled
112 235 141 284
OK

StatText Disabled
10 88 141 289
Serial Demo\0D\0D++
Shows Serial Port
Use\0D^0\0D^1\0D^2\0D^3

PicItem Disabled
10 10 96 81
128

Type ICN# = GNRL
  ,128 (0)
.H
0001 0000 0002 8000 0004 4000 0008 2000
0010 1000 0021 C800 0040 C400 0081 4200
0102 0100 0204 0080 0408 0040 0810 0020
1020 0010 2040 0008 4280 3F04 8300 4082
4381 8041 2003 3022 1005 C814 080E 7F8F
0402 3005 0201 0007 0100 8005 0080 6007
0040 1FE5 0020 021F 0010 0407 0008 0800
0004 1000 0002 2000 0001 4000 0000 8000
0001 0000 0003 8000 0007 C000 000F E000
001F F000 003F F800 007F FC00 00FF FE00
01FF FF00 03FF FF80 07FF FFC0 0FFF FFE0
1FFF FFF0 3FFF FFF8 7FFF FFFC FFFF FFFE
7FFF FFFF 3FFF FFFE 1FFF FFFC 0FFF FFFF
07FF FFFF 03FF FFFF 01FF FFFF 00FF FFFF
007F FFFF 003F FE1F 001F FC07 000F F800
0007 F000 0003 E000 0001 C000 0000 8000


TYPE PICT = GNRL
   ,128
.I
891
195 254 281 325
.H
1101 A000 82A0 008E 0100 0A00 0000 0002
D002 4098 000A 00C3 00F8 00FF 0148 00C3
00FE 00FF 0145 00C3 00FE 00FF 0145 0000
02F7 0002 F700 02F7 0002 F700 02F7 0002
F700 02F7 0002 F700 02F7 0002 F700 02F7
0006 FD00 000E FC00 07FD 0001 1F80 FD00
07FD 0001 7FC0 FD00 07FD 0001 FFF0 FD00
08FE 0002 03FF FCFD 0008 FE00 0207 FFFE
FD00 09FE 0003 1FFF FF80 FE00 09FE 0003
3FFF FFE0 FE00 09FE 0003 7FFF FFF8 FE00
0A02 0000 01FE FF00 FCFE 0008 0200 0003
FDFF FE00 0A02 0000 0FFD FF00 C0FF 000B
0700 001F FFFF 3FFF E0FF 000B 0700 007F
FFFE 1FFF F8FF 000B 0700 00FF FFFE 1FFF
FCFF 000B 0100 01FE FF02 27FF FCFF 000B
0100 01FE FF02 F9FF F8FF 000B 0100 00FE
FF02 FE7F F0FF 000B 0200 003F FEFF 019F
E0FF 000B 0200 001F FEFF 01E7 C0FF 000B
0200 003F FEFF 01F9 80FF 000B 0200 0033
FEFF 01FE 80FF 000A 0200 0060 FDFF 00C0
FF00 0B07 0000 607F FFFF FCC0 FF00 0B07
0000 601F FFFF F870 FF00 0B07 0000 6007
FFFF F0F8 FF00 0B07 0000 6001 FFFF F0F8
FF00 0B07 0000 6000 FFFF F0F8 FF00 0B07
0000 6038 3FFF B050 FF00 0A06 0000 607C
0FFF 30FE 000B 0700 0060 F603 FE30 A8FF
000B 0700 0060 E301 FC30 50FF 000B 0700
0060 C000 7830 20FF 000B 0700 0060 0000
1030 88FF 000B 0200 0060 FE00 0130 50FF
000A 0200 0060 FE00 0030 FE00 0B02 0000
60FE 0001 30A8 FF00 0B07 0000 6807 0700
B050 FF00 0A06 0000 681F 8FC0 B0FE 000B
0700 006C 7FDF F1B0 A8FF 000A 0200 0067
FEFF 0030 FE00 0B09 0000 63FF FFFE 31F4
1000 0B09 0000 307F DFF0 6046 3000 0B09
0000 381F 8FC0 E045 5000 0B09 0000 1C00
0001 C044 9000 0B09 0000 0E00 0003 8044
1000 0802 0000 07FE FFFD 0009 0500 0001
FFFF FCFD 0008 FE00 0280 0004 FD00 9800
0A00 FF00 F801 1901 4800 FF00 FE01 1901
4500 FF00 FE01 1901 4500 0008 FE00 0280
0004 FD00 08FE 0002 FFFF FCFD 0008 0200
0001 FEAA FD00 0802 0000 03FE 55FD 000A
0600 0006 FEAF EA80 FE00 0A06 0000 0D83
5835 40FE 000A 0600 001B 01B0 1AA0 FE00
0A06 0000 3501 5015 50FE 000A 0600 006A
82A8 2AA8 FE00 0A06 0000 D57D 57D7 F4FE
000A 0600 01AF AAFA AC1A FE00 0A06 0003
5055 0558 0DFE 000B 0700 06A0 2A02 A80A
80FF 000B 0700 0D60 3603 5415 40FF 000B
0700 0AB0 6B06 ABEA C0FF 000B 0700 0D5F
D5FD 5555 40FF 0009 0100 0AFC AA00 C0FF
0009 0100 0DFC 5500 40FF 0009 0100 0FFC
FF00 C0FF 0002 F700 02F7 0002 F700 02F7
0002 F700 02F7 0002 F700 A000 8FA0 0083
FF
```

© The Definitive MacTutor, Vol. 4

Advanced Mac'ing

Menus as Objects in TML Pascal

David B. Curtis
Wyoming, MN
MacTutor Vol. 4 No. 6

[One of the limitations of LightSpeed Pascal and Turbo Pascal, despite their popularity, is that they don't support objects. TML Pascal does support the object type, which is similar to a procedure. In this article, David Curtis shows us how to use the TML Pascal Object type on menus to create an elegant way to manage menus, especially a custom menu definition like an icon menu, shown in figure 1. Unfortunately, Tom Leonard of TML has not bothered to keep MacTutor supplied with updates to TML Pascal in over a year, so I have been unable to check the compilation and linking of this article. However, the code looks clean and should not have any problems. It is our hope that TML will keep MacTutor and the rest of the development community better informed and supplied with timely updates to their Pascal. -Ed]

In some applications, a graphic menu provides a friendlier user interface than the standard text list. MacDraw's "Fill", "Lines", and "Pen" menus are good examples. They *show* you the patterns you can choose from, rather than describe them in words. Can you imagine trying to describe the SysPat list verbally? Pictures are much more friendly. I have a similar problem in an application I am working on. It is a simple circuit analysis program, and I wish to have a menu of circuit models for the user to choose from. A circuit element is much more easily drawn than described. The brute force approach to this problem is to make a procedure that draws each circuit model in the menu area. That works, but I dislike it. First of all, it is hard to maintain. Second, it is just plain hard work to build the first time. Finally, it is code that has little hope of ever being useful in another program.

In case you didn't notice, my objections to the brute force approach are motivated largely by sloth. How could I avoid excess work, both now and in the future? I went out in search of the lazy man's way to a widely applicable, easily maintainable graphic menu, and came up with the icon menu. The icon menu procedure uses icons from a resource file to present a palette of selections. The code is generic across any application, and the icons can be maintained easily with ResEdit. Icons turn out to be a very nice size for my circuit model pictures. I suspect that icons would be good for a lot of graphic menu applications. (How about an icon based Transfer menu?) Figure 1 shows a prototype circuit model menu, implemented as an Icon menu.

Implementation Overview

I implemented the icon menu as a TML Pascal UNIT. (Any code that might get used in another program in the future I automatically put into a UNIT.) The OBJECT feature of Pascal

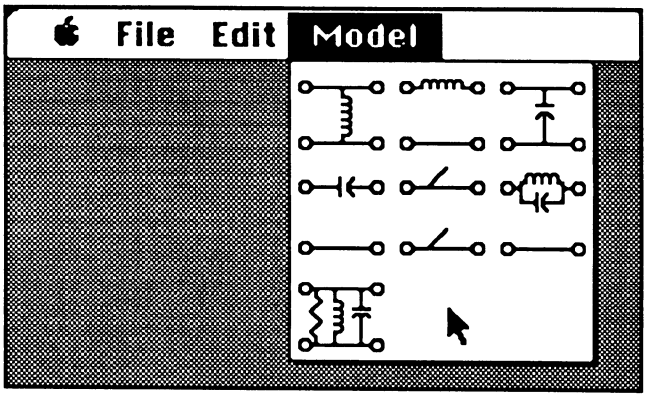


Fig. 1 Icon Menus as Objects in Object Pascal

is another powerful feature for reducing redundant work. I have used both features to create my icon menu, and usages of both are several layers deep. Each layer is quite simple, but some sort of road map to the structure is in order.

The lowest level unit is the trivial StdPoll ("standard poll") unit. I mention it only so that you don't wonder why I didn't. Eventually, I plan to put generic polling loop code in here, but in its present form, StdPoll's main purpose is to provide the global variable "Done", which gets set in response to a "Quit" menu choice. Now on to the interesting stuff.

StdMenu ("standard menu") is the key unit that everything is built upon. It defines a low level object called MenuHdlr ("menu handler"). A MenuHdlr object collects menu code into one place and "handles" all the associated behaviors. The two methods that a generic MenuHdlr has are Create, which should do whatever it takes to make a menu and insert it at the end of the menu bar; and Choose, which will be called to handle a menu choice. In the default case, Create just slurps up a normal menu resource from the resource fork. Choose, on the other hand, does nothing; it should be overridden in the application with appropriate code to handle the menu choices.

The StdMenu unit also provides pre-defined descendant objects of MenuHdlr, called AppleMenuHdlr, StdFileMenuHdlr, and StdEditMenuHdlr. These menu handlers provide minimal functionality for the Apple, File and Edit menus, respectively. The AppleMenuHdlr should be all most applications need to support a standard Apple menu. The Create method is overidden with code that pulls in desk accessory names, and the Choose method is overidden with code that displays an "About..." alert, or opens a desk accessory.

Notice that the AppleMenuHdlr has an additional method called "Setup", which does the additional pre-Create work of setting up an alert id for the About... selection. I have chosen the

convention of putting additional pre-create initialization for MenuHdlrs (if needed) in a Setup method, but Setup is not a method in the generic MenuHdlr. Why? Because every descendant object that overrides a method must have the same parameter list, and that is exactly what we *don't* want for a Setup method; we want the Setup method to be specific to the needs of a subspecies of MenuHdlr.

StdFileMenuHdlr and StdEditMenuHdlr exist in StdMenu only to provide bare minimum "Choose" functionallity. Most applications will want to override this code, but the minimal version is handy while prototyping other chunks of the application.

The program TestMenus uses AppleMenuHdlr, StdEditMenuHdlr and StdFileMenuHdlr. Look at the menu objects declared in the "var" section, and then look at the the procedures "InitApplication" and "MenuChoice" to see how the StdMenu unit is used. Notice how clean MenuChoice is because of the overridden Choose methods.

The IconMenu Unit

The meaty unit, as far as we are concerned, is IconMenu. This unit provides icon menu functionallity as a natural extension of the StdMenu facilities. The object type IconMenuHdlr, a descendant of MenuHdlr, adds a Setup method which is used to define the parameters of the icon menu. That includes what the first icon's resource id is (the rest must follow in sequence), how many icons you have to display, how many icons wide you want the menu, how many extra pixels of blank space you want around each icon, and what the name of the menu is. Setup calculates how many icons tall the menu needs to be. These parameters are then stored in the IconMenuHdlr data record.

The Create method is in charge of a couple of tricks. Since the IconMenu is a non-standard menu, we have to set up our own menu definition procedure. My Icon menu definition procedure is IconMenuDef, a global procedure in the implementation part of the unit. IconMenuDef is installed as the menu's definition procedure by the Create method. When the Create method first calls NewMenu, it gets a standard MenuInfo record. Since we want to store the larger IconMenuInfo record, Create calls SetHandleSize to "stretch" the storage block to the required size. The call to NewMenu also fills in the menuProc field with a handle to the standard menu definition procedure; we want to write over that with a handle to our own menu definition procedure. Be careful! The default menuProc handle that you get points to the one and only master pointer to the standard menu definition procedure. Assign a new handle to menuProc (which gets a new master pointer) before going through menuProc to set up a new menu definition. If you don't, you clobber *all* menus. I am embarrassed to say I learned that the hard way, although it *is* obvious when you stop and think about it.

The Create method also sets up a reference *in the menu record* to the IconMenuHdlr object. That way IconMenuDef can find its way back to the data telling it what icons to draw where. (In a normal menu, the item information itself is stored following the MenuInfo record.) To set up the object reference, the menuHandle is type converted to an IconMenuHandle, and SELF is tucked away in the IconMenuInfo record. SELF is a reference to the object calling the method. Each instance of an IconMenuHdlr must call Create, and thus each icon menu gets its own list of icons and its own height and width. The IconMenuInfo record defined in the unit interface provides the necessary type information for the Create method to place the object reference in the menu data.

The MDEF Procedure

MDEF procedures have been described before in MacTutor, also reprinted in The Complete MacTutor, Vol. 2, p. 248, 251); I will just touch on the basics here. If you need more information, dig out those back issues, or see IM Vol. I, p. 362.

A menu definition procedure must respond to three kinds of "messages", called mSizeMsg, mDrawMsg, and mChooseMsg. When the menu manager sends you the mSizeMsg, he wants to know how much space your menu takes on the screen. The menu manager sends you the mDrawMsg when he wants you to draw your menu on the screen; he has already saved the stuff underneath and given you a nice, white drawing space. When you get the mChoseMsg you should check the mouse location that you are given and do any hiliting and unhiliting necessary to behave like a menu. In a nutshell, that's all there is to it; but I will admit it took me a while to get all that to work.

My menu definition procedure has four local procedures. One for each menu manager message type, and one that returns a rectangle in global coordinates when given the number of a menu selection. This structure is borrowed straight from Daryl Lovato's MDEF example in the *TML Source Code Library*. (If I can make a small digression at this point, I would like to recommend the *Source Code Library* as a wonderful learning tool. Reading and modifying *Source Code Library* programs has taught me a lot about the Mac and its mysteries.)

DoSizeMsg and DoChoseMsg are both quite simple. All DoSizeMsg has to do is allow room for the necessary number of icons, plus the extra blank space in between. DoChoseMsg just searches for the icon that the cursor is over, and updates hiliting as necessary.

DoDrawMsg is a little more interesting, but still quite simple because we can make use of the icon support routines in the toolbox. This is where laziness really pays off; drawing all my circuit models through brute force code would have made this routine a nightmare. I suppose it would be good form to pre-load the icon resources for the menu, since a user could potentially end up in swap-a-floppy-land the way DoDrawMsg is now implemented. The Create method could easily preload icon resources, or the preload bit could be set with ResEdit.

Conclusion

The fanatical user might make some performance improvements to the IconMenuDef procedure. The item rectangles are

calculated now, that could be sped up with look-up techniques. The item search itself could be sped up by doing something more elegant than a linear search. The icon menu seems quite fast to me as it is, however, so I don't plan to fiddle with it. We non-fanatics need only zap up a few icons with ResEdit to have graphic menus.

```
{$L TestMenus/Rsrc }

program TestMenus;

uses
    Macintf, StdPoll, StdMenu, IconMenu;

const
    { Menu IDs }
    AppleMenuID = 256;
        AboutAlertID = 1000;
    FileMenuID = 257;
    EditMenuID = 258;
    CktMenuID = 259; { not a resource id }
        FirstCktIcon = 500;
        NumCktIcons = 7;
        IconsWide = 3;
        BfSpace = 3;

var
    { Instances of Menu Handlers }
    AppleMenu : AppleMenuHdlr;
    FileMenu : StdFileMenuHdlr;
    EditMenu : StdEditMenuHdlr;
    CktMenu : IconMenuHdlr;

    { Event polling variables }
    TheEvent : EventRecord;
    EventIsForMe : Boolean;

{
|   DoMenuChoice - do a menu selection
}
procedure DoMenuChoice (MenuCode : LongInt);
begin
    case HiWord(MenuCode) of
        AppleMenuID: AppleMenu.Choose (LoWord(MenuCode));
        FileMenuID : FileMenu.Choose (LoWord(MenuCode));
        EditMenuID : EditMenu.Choose (LoWord(MenuCode));
        CktMenuID: CktMenu.Choose (LoWord(MenuCode)) end;
    HiliteMenu (0) end;

{
|   PollEvent - check event queue and dispatch event, if any
}
procedure PollEvent;
var
  TempWindow : WindowPtr;
begin
  EventIsForMe := GetNextEvent(everyEvent,TheEvent);
  if EventIsForMe
  then case TheEvent.what of
     mouseDown : case FindWindow(TheEvent.where,TempWindow) of
         inMenuBar: DoMenuChoice(MenuSelect(TheEvent.where));
         inSysWindow : SystemClick (TheEvent, TempWindow);
         otherwise { ignore } end;
     keyDown : if BitAnd(TheEvent.modifiers,CmdKey) <> 0
         then DoMenuChoice(MenuKey(CHR(BitAnd(
TheEvent.message,CharCodeMask))));
  otherwise { ignore } end end;

{
|   InitApplication
}
procedure InitApplication;
begin
    Done := False;

    { Create the menus }
    New (AppleMenu);
    AppleMenu.Setup (AboutAlertID);
    AppleMenu.Create (AppleMenuID);

    New (FileMenu);
    FileMenu.Create (FileMenuID);

    New (EditMenu);
    EditMenu.Create (EditMenuID);

    New (CktMenu);
    CktMenu.Setup (FirstCktIcon, NumCktIcons, IconsWide,
            BfSpace, 'Model');
    CktMenu.Create (CktMenuID);

    DrawMenuBar;

    InitCursor end;

{
|   Main
}
begin
  InitTheMac;
  InitApplication;
  repeat
    SystemTask;
    PollEvent;
    until done end.


{$A+} { Interlinear source on compile to .asm }
{$R+} { Range checking }

unit StdPoll;

INTERFACE

uses
    MacIntf;

var
    Done : Boolean; { program is all done }

procedure InitTheMac;

IMPLEMENTATION

procedure InitTheMac;
begin
    InitGraf (@ThePort);
    InitFonts;
    InitWindows;
    InitMenus;
    TEInit;
    InitDialogs (NIL) end;
end.

{
|   StdMenu provides standard menu objects.
}
unit StdMenu;

interface

uses MacIntf, StdPoll;

const
```

```
    { Standard File menu item numbers }
    NewItem = 1;
    OpenItem = 2;
    CloseItem = 3;
    { --- }
    SaveItem = 5;
    SaveAsItem = 6;
    RevertItem = 7;
    { --- }
    PageSetupItem = 9;
    PrintItem = 10;
    { --- }
    QuitItem = 12;

type
    {
    |   MenuHdlr is the base object for all menus.
    |   The Create method installs the menu at the end of
    |   the menu bar.  The Choose method should be overridden
    |   with code to perform a menu choice.
    }
    MenuHdlr = object
        TheMenu : MenuHandle;
        procedure Create (RsrcID : Integer);
        procedure Choose (Choice : Integer);
    end;

    {
    |   AppleMenuHdlr provides all the functionallity of
    |   a standard Apple menu.  It must be Setup with an
    |   alert id for the About… message before Create is
    |   called.
    }
    AppleMenuHdlr = object (MenuHdlr)
        AboutID : Integer;
        procedure Create (RsrcID : Integer); override;
        procedure Choose (Choice : Integer); override;
        procedure Setup (AlertID : Integer);
    end;

    {
    |   StdFileMenuHdlr is a minimal File menu.  The Choose
    |   method can Close DA's and Quit.
    }
    StdFileMenuHdlr = object (MenuHdlr)
        procedure Choose (Choice : Integer); override;
    end;

    {
    |   StdEditMenuHdlr is a minimal Edit menu.  The Choose
    |   method supports DA editing.
    }
    StdEditMenuHdlr = object (MenuHdlr)
        procedure Choose (Choice : Integer); override;
    end;

implementation

{
|   Generic Menu Handler
}
procedure MenuHdlr.Create (RsrcID : Integer);
begin
    { Read in a standard menu resource.
    | Remember: Object variables are Handles!
    |'Self' is set only
    | as you enter the method, so HLock it if you call any
    | potential heap-scramblers! }
    HLock(Handle(Self));
    TheMenu := GetMenu (RsrcID);
    InsertMenu (TheMenu, 0);
    HUnlock(Handle(Self)) end;

procedure MenuHdlr.Choose (Choice : Integer);
begin end;

{
|   Apple Menu Handler
}
procedure AppleMenuHdlr.Create (RsrcID : Integer);
begin
    { Read in the Apple menu stub, and add DA's }
    HLock(Handle(Self));
    TheMenu := GetMenu (RsrcID);
    AddResMenu (TheMenu, 'DRVR');
    InsertMenu (TheMenu, 0);
    HUnlock(Handle(Self)) end;

procedure AppleMenuHdlr.Choose (Choice : Integer);
var
    AccName : Str255;
    AccNumber : integer;
begin
    { Post alert or open DA }
    if Choice = 1
    then AccNumber := Alert (AboutID, NIL)
    else begin
        GetItem (TheMenu, Choice, AccName);
        AccNumber := OpenDeskAcc (AccName) end end;

procedure AppleMenuHdlr.Setup (AlertID : Integer);
begin
    AboutID := AlertID end;

{
|   Standard File Menu
}
procedure StdFileMenuHdlr.Choose (Choice : Integer);
var
    FrontWP : windowPeek;
begin
    case Choice of
    NewItem : ;
    OpenItem : ;
    CloseItem : begin { If frontmost window DA, close it. }
        FrontWP := windowPeek(frontWindow);
        if FrontWP^.windowKind < 0
        then CloseDeskAcc(FrontWP^.windowKind) end;
    SaveItem : ;
    SaveAsItem : ;
    RevertItem : ;
    PageSetupItem : ;
    PrintItem : ;
    QuitItem : Done := true;
    otherwise  { nothing } end end;

{
|   Standard Edit menu
}
procedure StdEditMenuHdlr.Choose (Choice : Integer);
var
    Trash : Boolean;
begin
    Trash := SystemEdit (Choice-1) end;

end. { of StdMenu unit }

{
|   Icon menu MDEF unit
|
|   Creates and manages a menu of Icons.
}
unit IconMenu;

interface
```

```
uses
    Macintf, StdPoll, StdMenu;

type
    IconMenuHdlr = object (MenuHdlr)
        StartIcon,    { The resource ID of first icon }
        NumIcons,     { How many icons to use (ID's in
                      sequence) }
        IconsWide,    { Menu shape... }
        IconsTall,    { More menu shape. }
        BufferSpace   { Extra white pixels around each icon }
            : Integer;
        Title         { The name of the menu }
            : Str255;

        procedure Create (RsrcID : Integer); override;
        procedure Setup (
            StartIconReq,
            NumIconsReq,
            IconsWideReq,
            BufferSpaceReq
                : Integer;
            TitleReq
                : Str255); end;

    IconMenuPtr = ^IconMenuInfo;
    IconMenuHandle = ^ IconMenuPtr;

    IconMenuInfo = record
        StdStuff : MenuInfo; { The default MenuInfo record }
        Handler : IconMenuHdlr { Reference to the above
                                 handler object }
            end;

implementation

const
    IconSize = 32;    { How big is an icon }

{
|   IconMenuDef is installed as the MDEF procedure.
|   See IM Vol. I, p. 362
}
procedure IconMenuDef (
    Message : Integer;
    SelectedMenu : IconMenuHandle;
    var MenuRect : Rect;
    HitPt : Point;
    var WhichItem : Integer);

    {
    |   ItemRect - function to find the rectangle ( in global
    |   coordinates) of a given item number.
    }
    function ItemRect (
        ItemNum : Integer;
        MenuRect : Rect;
        SelectedMenu : IconMenuHandle) : Rect;

    var
        TempRect
            : Rect;
        ItemLess1,
        ItemSize
            : Integer;

    begin
        { If ItemNum is a real item, then return the }
        { global coordinates of the item's rectangle; }
        { otherwise return empty rect. }
        if (ItemNum >= 1)
        and (ItemNum <= SelectedMenu^^.Handler.NumIcons)
        then with SelectedMenu^^.Handler do begin
            ItemLess1 := ItemNum - 1;
            ItemSize := IconSize + 2 * BufferSpace;
            TempRect.top := (ItemLess1 div IconsWide)
                    * ItemSize + MenuRect.top;
            TempRect.left := (ItemLess1 mod IconsWide)
                    * ItemSize + MenuRect.left;
            TempRect.bottom := TempRect.top + ItemSize;
            TempRect.right := TempRect.left + ItemSize end
        else begin
            TempRect.top := 0;
            TempRect.left := 0;
            TempRect.bottom := 0;
            TempRect.right := 0 end;
        ItemRect := TempRect end;

    {
    |   DoDrawMessage - handle the menu manager Draw command
    }
    procedure DoDrawMessage (
        SelectedMenu : IconMenuHandle;
        MenuRect : Rect);

    var
        Selection : Integer; { Current selection }
        SelRect : Rect;   { Current selection's Rect }
        TheIcon : Handle; { Handle to selection's Icon }

  begin
     { Get every icon in the menu and plot it. }
     HLock(Handle(SelectedMenu));
     with SelectedMenu^^.Handler do begin
  for Selection := 1 to NumIcons do begin
            SelRect := ItemRect(Selection, MenuRect, Selected-
Menu);
            InsetRect(SelRect,BufferSpace,BufferSpace);
            TheIcon := GetIcon (StartIcon + Selection - 1);
            PlotIcon (SelRect, TheIcon) end end;
     HUnlock(Handle(SelectedMenu)) end;

    {
    |   DoChooseMessage - handle menu manager Choose command
    }
    function DoChooseMessage (
        SelectedMenu : IconMenuHandle;
        MenuRect : Rect;
        HitPoint : Point;
        OldSelection : Integer) : Integer;
    var
        SelRect : Rect;
        Found : boolean;
        Selection : Integer;
        OldSelRect : Rect;
  begin
     Selection := 1;
     Found:= false;
     { Find out which item mouse is over, if any. }
     repeat
  SelRect := ItemRect (Selection, MenuRect, SelectedMenu);
        Found := PtInRect (HitPoint,SelRect);
        if not Found then Selection := Selection + 1;
     until ((Selection > (SelectedMenu^^.Handler.NumIcons))
        or (Found));
     { Update hiliting as necessary }
     if Found
     then begin { in an item }
        if (Selection <> OldSelection)
        then begin { in a different item, change hiliting }
           OldSelRect := ItemRect(OldSelection,
              MenuRect, SelectedMenu);
           InvertRect(OldSelRect);
           InvertRect(SelRect) end;
        DoChooseMessage := Selection end
  else begin { not in a item, unhilite old }
```

```
    OldSelRect := ItemRect(OldSelection, MenuRect, Selected-
Menu);
        InvertRect (OldSelRect);
           DoChooseMessage := 0 end end;

    {
    |   DoSizeMessage - handle the menu manager Size command
    }
  procedure DoSizeMessage (
     var Menu : IconMenuHandle);
  begin
  with Menu^^.Handler do begin
  Menu^^.StdStuff.menuWidth:= IconsWide*(IconSize + 2 *
BufferSpace);
  Menu^^.StdStuff.menuHeight:= IconsTall*(IconSize + 2 *
BufferSpace)
        end end;

{
| IconMenuDef - main
}
begin
  case message of
     mSizeMsg : DoSizeMessage (SelectedMenu);
  mDrawMsg : DoDrawMessage (SelectedMenu, MenuRect);
        mChooseMsg : WhichItem := DoChooseMessage (
           SelectedMenu,MenuRect,HitPt,WhichItem) end end;

function Min (a,b : Integer) : Integer;
begin
    if a < b
    then Min := a
    else Min := b end;

procedure IconMenuHdlr.Setup (
             StartIconReq,
             NumIconsReq,
             IconsWideReq,
             BufferSpaceReq
                  : Integer;
             TitleReq
                  : Str255);
begin
    StartIcon := StartIconReq;
    NumIcons := NumIconsReq;
    IconsWide := IconsWideReq;
    { Calculate IconsTall from NumIcons and IconsWide }
    IconsTall := NumIcons div IconsWide
            + Min (NumIcons mod IconsWide, 1);
    BufferSpace := BufferSpaceReq;
    Title := TitleReq end;

procedure IconMenuHdlr.Create (RsrcID : Integer);
{ RsrcID isn't a resource ID in this case, just a menu id }
var
    TrickyHandle : IconMenuHandle; { Used for type coercion }

begin
    HLock(Handle(Self));
    TheMenu := NewMenu (RsrcID, Title);
    { Get a plain menu record }

    SetHandleSize (Handle(TheMenu), SizeOf(IconMenuInfo));
    { Stretch it }
    if MemError = 0
    then begin
        { Assign the icon MDEF proc }
        TheMenu^^.menuProc := NewHandle(0); {Important! }
        TheMenu^^.menuProc^ := @IconMenuDef;
        { Stuff in reference to the MenuHandler object }
        TrickyHandle := IconMenuHandle (TheMenu);
        TrickyHandle^^.Handler := SELF;
        { Insert at end of menu bar }
        CalcMenuSize (TheMenu);
        InsertMenu (TheMenu,0) end
    else
        SysBeep (1);
    HUnlock(Handle(Self)) end;

end. { of IconMenu unit }
```

```
!PAS$Xfer

/Globals -4
TestMenus
PAS$Library
macintf
stdpoll
stdmenu
iconmenu


* IconMenu.R resources
*

IconMenu.rsrc
????????

Type EEIN = STR
  ,0     ;;0 by convention
Icons in Menus by David Curtis \A9 MacTutor 1988.

Type BNDL
  ,128
  EEIN  0
  ICN# 1
  0 128
  FREF 1
  0 128

Type FREF
  ,128
  APPL  0    ;; local id 0 for icon list

* —— Multifinder ——

Type SIZE = GNRL
    ,-1
.I
16384    ;; $4000 = bit 14 set
.L
148000   ;;  recomended
.L
128000   ;;  minimum


Type ALRT
  ,1000 (0)
60 128 260 368
100
4444

Type DITL
  ,100 (0)
2

Button
133 128 153 188
OK

staticText
60 41 80 159
About Menu Test
```

```
Type ICON = GNRL
  ,505 (0)
.H
0000 0000 0000 0000 0036 D800 0049 2400
0049 2400 71C9 270E 8900 0111 8F00 01F1
8900 0111 7104 410E 0104 8100 0105 0100
01FD FF00 0005 0000 0004 8000 0004 4000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 7000 000E
8800 0011 8FFF FFF1 8800 0011 7000 000E

Type ICON = GNRL
  ,504 (0)
.H
0000 4000 0000 8000 0001 0000 0002 0000
0004 0000 7008 000E 8810 0011 8FFF FFF1
8800 0011 7000 000E 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 4000 0000 8000
0001 0000 0002 0000 0004 0000 7008 000E
8810 0011 8FFF FFF1 8800 0011 7000 000E

Type ICON = GNRL
  ,503 (0)
.H
0000 0000 0000 0000 0000 0000 0000 0000
0001 0800 7001 100E 8801 2011 8FFF 3FF1
8801 2011 7001 100E 0001 0800 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 7000 000E
8800 0011 8FFF FFF1 8800 0011 7000 000E

Type ICON = GNRL
  ,506 (0)
.H
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 7000 000E 8800 0011 8FFF FFF1
8A04 0111 7204 010E 0204 0100 0104 0100
0087 0100 0100 8100 0200 8100 0407 0FE0
0800 8000 0400 87C0 0207 0920 0100 8100
0080 8100 0107 0100 0200 8100 0400 8100
0807 0100 0404 0100 0204 0100 7204 010E
8A04 0111 8FFF FFF1 8800 0011 7000 000E

Type ICON = GNRL
  ,501 (0)
.H
0000 0000 0000 0000 0000 0000 0000 0000
006D B600 7092 490E 8892 4911 8F92 49F1
8800 0011 7000 000E 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 7000 000E
8800 0011 8FFF FFF1 8800 0011 7000 000E


Type ICON = GNRL
  ,500 (0)
.H
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 7000 000E 8800 0011 8FFF FFF1
8800 8011 7000 800E 0000 8000 0000 E000
0000 1000 0000 1000 0000 E000 0000 1000
0000 1000 0000 E000 0000 1000 0000 1000
0000 E000 0000 1000 0000 1000 0000 E000
0000 1000 0000 1000 0000 E000 7000 800E
8800 8011 8FFF FFF1 8800 0011 7000 000E
```

```
Type ICON = GNRL
  ,502 (0)
.H
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 7000 000E 8800 0011 8FFF FFF1
8800 8011 7000 800E 0000 8000 0000 8000
0000 8000 0000 8000 0007 F000 0000 0000
0000 0000 0001 C000 0002 A000 0004 9000
0000 8000 0000 8000 0000 8000 0000 8000
0000 8000 0000 8000 0000 8000 7000 800E
8800 8011 8FFF FFF1 8800 0011 7000 000E

Type MENU
  ,256 (0)
\14
About Test Menus…
(-‹

Type MENU
  ,257 (0)
File
(New/N
(Open
Close
(-
(Save
(Save As…
(Revert…
(-
(Page Setup…
(Print…
(-‹
Quit/Q

Type MENU
  ,258 (0)
Edit
Undo
(-‹
Cut/X
Copy/C
Paste/V
(-
Clear

Type ICN# = GNRL
  ,128 (4)  ;; The Appl. Icon
.H
003FFC0 000400200 00800100 01000080
0200004 004000020 09FFFFD0 13ABF568
23FFFFE 440422082 807E2081 80423F81
807E208 18000208 1 80003F81 80002081
801E208 1800E3F81 800E2081 801A2081
80303F8 180602081 40C02082 21803F84
1300208 808003F90 04000020 02000040
0100008 000800100 00400200 003FFC00
*
003FFC0 0007FFE00 00FFFF00 01FFFF80
03FFFFC 007FFFFE0 0FFFFFF0 1FFFFFF8
3FFFFFF C7FFFFFFE FFFFFFFF FFFFFFFF
FFFFFFF FFFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFF FFFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFF FFFFFFFFF 7FFFFFFE 3FFFFFFC
1FFFFFF 80FFFFFF0 07FFFFE0 03FFFFC0
01FFFF8 000FFFF00 007FFE00 003FFC00
```

Database Corner

4th Dimension® Externals

Todd Carper
ACIUS, Inc.
Cupertino, CA
MacTutor Vol. 4 No. 9

[Todd Carper is a Technical Support Engineer for 4th Dimension® and handles all external procedure development. He has been there for one year.]

External Procedures in 4th Dimension®

4th Dimension is a powerful and sophisticated database for the Macintosh. It provides an excellent environment for developing custom databases and applications. 4th Dimension allows you to quickly generate screens, menus, buttons, scrollable areas, and procedures to create a true Macintosh interface without starting from scratch.

4th Dimension's language enables you to avoid pages of code and hours of work by accessing the Macintosh ToolBox with high level commands. Though the language is very powerful it may not contain all the built-in commands to meet every need. For this reason 4th Dimension was designed as an open architecture database that can utilize your own commands. These commands are called "external procedures", and this document will discuss the design and implementation of external procedures.

External procedures enable you to add unlimited functionality to a 4th Dimension database or application. For example, you can add sound, pop-up menus, picture buttons, mathematics, or telecommunications capabilities. External procedures are extensions to the 4th Dimension commands that you create like XCMDs for HyperCard. Once the procedure is installed, it is part of 4th Dimension and can be called just as you call built-in commands.

External procedures can be written in assembly language, or any higher level language that compiles to native 68000 code. It is very important that the code be stand-alone, in that it does not require additional modules to function properly. This article will use Pascal programs created with the Apple Macintosh Programmers Workshop (MPW) Pascal.

Calling the Externals

When 4th Dimension 'calls' the external procedure, it simply performs a direct link, Link A6, to the routine and executes the supplied code until the code ends, and by nature executes a return statement to the calling routine.

Because the supplied code is being called from within another application, you should not try to access any global variables, as those are stored off the A5 register and it's contents may not be correct for your call. Therefore always declare your variables local. If you need the variables to be used as if they were global then define them within a procedure, and have all called procedures and functions defined within the procedure defining the 'global' variables.

The idea is to make all references relative. This is important because when the code is moved into 4th Dimension, the Code 0 segment is not used. (This is where the Jump table is located). Therefore if you have procedures you want to call, they should be declared as FORWARD and your top most procedure should be the one that 'runs' your program.

Following is an example external procedure. This procedure will receive a number, multiply the number by 2, and return the result. Because the variable is declared as a var in the procedure parameter list, the result can be returned to 4th Dimension.

```
Program Ext_Doubler;
Var   DumInt:Integer;

procedure Doubler(var TheInt:Integer);
begin
  TheInt:=TheInt*2;
end; (Doubler)

Begin
  Doubler(DumInt); (Used for compile only)
End.   (Main Block)
```

Following is an example utilizing multiple procedures which doesn't take any parameters. It simply assigns the string 'Hello there' to the global variable Global2. It then beeps once for each character in the string. The variables Global1 and Global2, appear to be global type variables because the procedures which use them are within the scope of the procedure which defines both the variables and the procedure using the variables.

The main block calls the procedure Caller, (this call is made for compiler considerations only.) Caller then calls the procedure to be executed, HaveGlobals. HaveGlobals was declared as FORWARD so that Caller would know that it exists. HaveGlobals defines the variables to be used by the procedures defined in it's scope. HaveGlobals assigns the string to Global2. It then calls First to assign the length of Global1 to Global2. Then Second is called to have the system beep Global1 times.

```
Program TestProg;

Uses Memtypes, QuickDraw, OSIntf;

procedure HaveGlobals; FORWARD;
(so Caller knows about the procedure)
```


© The Definitive MacTutor, Vol. 4

```
procedure Caller;
begin
  HaveGlobals;
end;

procedure HaveGlobals;
var
  Global1:Integer;
  Global2:str255;

    procedure First;
    begin
      Global1:=Length(Global2);
    end; (First)

    procedure Second;
    var Count:Integer;
    begin
      For Count:=1 to Global1 do Sysbeep(2);
    end; (Second)

begin (HaveGlobals)
  Global2:='Hello there';
  First;
  Second;
end; (HaveGlobals)

Begin (Main Block)
  Caller;
End. (program)
```

Creation of External procedures

Be sure that if your external compiles to more than 1 code segment, or if one of the linked modules is placed in it's own segment, that you join the segments into 1 complete segment. This is important if your application uses the SANE library.

You can compile to an application and then use the 4D External Mover™ to transfer the code segment and to specify the parameters. You can also compile directly to a Proc.Ext type file and specify the parameter list. Then you can use the 4D External Mover to copy the already set-up external to the proper Proc.Ext file. There will also be an item in the list, (if viewed in the 4D External Mover), with the name %A5Init. This item may be deleted from the list as it is not needed by the external.

Here is an example of a MPW linker spec to join the SANE code segment with the Main code segment and the method of installing directly into a Proc.Ext type file.

```
link -w -p MyProg.p.o ∂
  "{Libraries}"Interface.o ∂
  "{Libraries}"Runtime.o ∂
  "{PLibraries}"Paslib.o ∂
  "{PLibraries}"SANELib.o ∂
  -rt '4DEX=15000'∂  #Specify resource ID.
  # Joins segment Main and the Sane libraries into
  # one. *
  -sg "Main2=Main,SANELib "∂
  # Change segment name. **
  -sn 'Main2'='MyProg(&S;&R)'∂
  # Specify the creator and type.
  -c "4DMX" -t "PEXT"∂
    -o MyProg
```

*Note: for this external procedure, the Sane libraries are needed, therefore, they must exist in the same segment as the main code. If your code does not require the Sane libraries then this command is not necessary. Be sure to note the case used in "SANELib " and the space which is between 'b' and the quote mark.

**The segment name was changed from Main2 to MyProg, the additional parameters, in parens, specify the types of the parameters being passed from 4th Dimension. In this example, (&S;&R), specifies that a String and a Real are to be passed. String first, Real second. Each parameter is preceded by an ampersand, '&'. A semicolon is used to separate each parameter.

The parameter types are as follows:

I	Integer
L	LongInt
R	Real
S	String
T	Text
D	Date
P	Picture

Following are the Pascal type definitions for Date and Text fields.

```
    Date     type
                                        Date4D = record
day:integer;
month:integer;
year:integer;
                                        end;
    Text     type
                                        TERec4D = record
length:integer;
text:CharsHandle;
                                        end;
```

With the release of the 128k ROMs, the restriction of code segments being less than 32k has gone away. Because of this, your external procedure may be very large. There is one important note however; you cannot perform a branch to an address which is farther than 32k from where you are branching from. The linker will catch this and you will have to put in a 'stepping stone' procedure. All these procedures do is call the procedure you wanted to call originally. They are located between the calling procedure and the procedure to be called and are used as stepping stones to get to the proper procedure.

In this example, we want to call Proc4 from Proc1. Proc1

and Proc3 are very large; such that the distance from the call to Proc3 and the location of Proc3 is greater than 32k when compiled. To accomplish the call we need to insert a 'stepping stone' procedure after Proc1 called Step. Proc1 calls the procedure Step. Step then calls Proc3. In this way, the distance between calls is less than 32k.

```
Program Ext_StepStone;

Uses Memtypes;

Var
… {Variable definitions}

procedure Proc2; Forward;
procedure Proc3; Forward;
procedure Step; Forward;

procedure Proc1;
var
  MyVar:str255;
begin
  MyVar:='Hello';
  {Now we call Step to get to Proc3}
  Step;
  … {Lot's of Code}
end;   {Proc1}

procedure Step;
begin
  {This routine just calls Proc3}
  Proc3;
end;   {Step}

procedure Proc2;
begin
  … {Lot's of code}
end;   {Proc2}

procedure Proc3;
begin
  {Here's the code we want to use}
  … {Some more code}
end;   {Proc3}

Begin  {Main Block}
  Proc1;
End. {Main Block}
```

Handles and External procedures

Whenever storing a data structure, be sure that you are accessing it via a Handle and that you lock the Handle if you want to ensure that it is there when you return.

Be sure to Dispose of all Handles that you have created when you have finished with them. If you leave unused handles in memory, you could run out of memory at a later point in the program.

Using Pictures in Externals

When passing Picture variables, to and from external procedures, be sure to include room for the additional 6 bytes needed by 4th Dimension to store the (Horizontal, Vertical) origin of the picture used when it is displayed in an On-Background field, (4 bytes), and the mode in which the picture is to be displayed, (2 bytes). For most pictures this means using SetHandleSize to increase the allocated space:

```
    SetHandleSize(Handle(MyPic),
GetHandleSize(Handle(MyPic))+6)
```

* Remember, Pictures can now be greater than 32k in size. Therefore the PicSize element of the PicHandle record, may not be valid. You should always call GetHandleSize to learn the size of the Picture.

When using externals it is sometimes necessary to save a complex data structure or just a handle of a type unknown to 4th Dimension. To store types of handle other than PicHandle, you can simply use Type Coercion to change the handle to type PicHandle and then pass the handle back to 4th Dimension in the form of a picture. Do not attempt to display the 'picture' as strange things will definitely happen.

Here is an example of storing SND's in a 4th Dimension picture field. This example reads an SND resource from a file and returns to 4th Dimension a picture variable containing the SND. You can then store the picture (SND) and replay it at a later time using PlayPict, (the routine PlayPict follows this one). SoundToPict takes 3 parameters, SoundToPict(Parm1;Parm2;Parm3). Parm1 is the name of the file on disk containing the SND. Parm2 is the name of the SND to be retrieved. Parm3 returns a picture variable containing the SND. For information about the Resource Manager, see Inside Macintosh.

```
Program Ext_SoundToPict;

Uses MemTypes, QuickDraw, OSIntf,
    ToolIntf, PackIntf, Sound;

Var
  DFile, DSound:str255;
  DPict:PicHandle;

procedure SoundToPict(
  var TheFile, SoundName:str255;
  var MyPict:PicHandle);

type
  FFSynthHandle = ^FFSynthPtr;

var
  MySound:FFSynthHandle;
  TheRes:integer;
  MyPtr:Ptr;
  MyFFPtr:FFSynthPtr;

begin
  MyPict:=Nil;
  TheRes:=OpenResFile(TheFile);
  MySound:=FFSynthHandle(GetNamedResource
  ('Snd ',SoundName));  {Locate SND}
  {Test if the sound was found and read}
  if MySound<>Nil then
  begin
    Hlock(Handle(MySound));
```

```
      {Copy the Pointer}
      MyPtr:=Ptr(MySound^);
      {Type coerce the new Ptr}
      MyFFPtr:=FFSynthPtr(MyPtr);
      {Set the playback mode and ratio}
      MyFFPtr^.mode:=FFMode;
      MyFFPtr^.count:=FixRatio(1,1);
      MyFFPtr^.wavebytes[0]:=0;
      {Type coerce the handle to type}
      {PicHandle}
      MyPict:=Pichandle(MySound);
      {Lock the picture so we don't lose it}
      HLock(Handle(MyPict));
      {Coerce and Copy the pointer}
      MyPict^:=PicPtr(MyPtr);
      {Unlock the Handles before leaving}
      HUnlock(Handle(MySound));
      Hunlock(Handle(MyPict));
      {Now Detach the Resource since we}
      {have another handle to it.}
      DetachResource(Handle(MySound));
    end; {if mysound<>Nil}
    CloseResFile(TheRes);
  end;  {SoundToPict}

  Begin
    SoundToPict(DFile, DSound, DPict);
  End.  {Main Block}
```

This routine is used to play the SND that was stored in a 4th Dimension picture field. It takes a 4th Dimension picture variable and converts the 'picture' to a sound. The sound is then played with StartSound. If the passed picture variable does not contain information generated by SoundToPict, you will get unexpected results. For information about the Sound Driver, see Inside Macintosh.

```
Program Ext_PlayPict;

Uses MemTypes, QuickDraw, OSIntf,
     ToolIntf, PackIntf, Sound;

Var
  DPict:PicHandle;
  DFile, DSound:integer;

procedure PlayPict(
  var Numer, Denom:integer;
  var MyPict:PicHandle);

type
  FFSynthHandle=^FFSynthPtr;

var
  MySound:FFSynthHandle;
  MyPtr:Ptr;
  MyFFPtr:FFSynthPtr;

begin
  {make sure that we recieved something}
  if MyPict<>Nil then
  begin
    {Coerce the handle}
    MySound:=FFSynthHandle(MyPict);
    {Lock the sound so we don't lose it}
    HLock(Handle(MySound));
    {Copy the pointer}
    MyPtr:=Ptr(MySound^);
    {Type Coerce the pointer}
    MyFFPtr:=FFSynthPtr(MyPtr);
    {Set the playback mode}
    MyFFPtr^.mode:=FFMode;
    {Set the playback count to the}
    {parameters passed.}
    MyFFPtr^.count:=FixRatio(Numer, Denom);
    MyFFPtr^.wavebytes[0]:=0;
    {Play the sound}
    StartSound(MyPtr,
      GetHandleSize(Handle(MySound)),
      POINTER(-1));
    {Unlock the sound so it can be}
    {disposed}
    HUnLock(Handle(MySound));
  end;  {if MyPict<>Nil}
end;  {PlayPict}

Begin
  PlayPict(DFile, DSound, DPict);
End.  {Main Block}
```

The Grafport and External procedures

If you change the current Grafport or it's settings, you must restore the Grafport to it's original status before returning to 4th Dimension. For example, if you were to call an external procedure that changed the current Grafport to an off-screen port, perform some drawing operations, and use CopyBits to copy the image to the original Grafport, you must restore the Grafport to it's original location before you leave the external procedure. You cannot access QuickDraw globals, therefore you must do the following to track the current GrafPort:

```
{Get the current GrafPort, DO NOT use  ThePort}
GetPort(OrgPort);
{Set to the new port}
SetPort(MyPort);
.
.
.
{Set the port back to it's original status}
SetPort(OrgPort);
```

Changing the Mac environment

It is sometimes necessary to modify the settings of certain Mac items, such as the Serial Port buffer size. By default the buffer size is 64 bytes. Many applications that use the serial ports for communication need to have the buffer much larger. The buffer size can be modified while in 4th Dimension but you must remember to re-assign the buffer to it's original location before exiting 4th Dimension or else the new buffer might be cleared by another system operation and there would be an erroneous pointer to buffer space.

Here is an example of how to modify the serial port buffers. This routine creates a new buffer the size of NbOfBytes for serial port access. SetSerialBuf takes 3 parameters, SetSerialBuf(Device;BufHandle;BufSize). Where Device contains the value 0 or 1 to specify Printer or Modem, respectively. BufHandle will return the address of the generated buffer. (Do not change this variable after the call, as it is needed to find and

dispose of the buffer later.) BufSize specifies the size of the buffer in bytes, to be created. BufSize will return the size of the buffer actually created. The minimum buffer size is 1024 bytes. The maximum buffer size is 32000 bytes. In the case of an error, BufSize will return a negative number specifying the Macintosh Error Code. For an explanation of Macintosh Error Codes and information about the Device Manager, see Inside Macintosh.

```
Program Ext_SetSerBuf;

Uses MemTypes,QuickDraw,OSIntf,ToolIntf,
  Packintf;
{$D+}
{$R-}
Var
  BufHandle:Handle;
  PrinterOrModem,NbOfBytes:Integer;

procedure SetSerBuf(
  var PrinterOrModem:Integer;
  var BufHandle:Handle;
  var NbOfBytes:Integer);

var Error,RefNum:Integer;

begin
  {default buffersize of 1024 bytes}
  if NbOfBytes<1 then NbOfBytes:=1024;
  {max buffer size is 32000}
  if NbOfBytes>32000 then
    NbOfBytes:=32000;
  {generate space for the new buffer}
 BufHandle:=NewHandle(Ord4(NbOfBytes));
  {Test for Handle allocation error}
  Error:=MemError;
  if Error=NoErr then
  begin
    {Move Handle out of the way to avoid}
    {fragmentation.}
    MoveHHI(BufHandle);
    {Lock that puppy down}
    HLock(BufHandle);
    {Set Reference Number accordingly.}
    {0 is Printer, 1 is Modem}
    if PrinterOrModem=0 then
      RefNum:=-8 {Printer}
    else
      RefNum:=-6; {Modem}
    {Reassign the serial port buffer}
Error:=SerSetBuf(RefNum,Ptr(BufHandle^)
    ,Ord(GetHandleSize(BufHandle)));
    {Test for errors from reassignment}
    if Error<>NoErr then
    begin
      {If errors then beep and undo what we}
      {have done.}
      SysBeep(10);
      {Assign the error code to NbOfBytes}
      {so that it is returned}
      NbOfBytes:=Error;
      {Unlock and dispose of buffer}
      HUnLock(BufHandle);
DisposHandle(BufHandle);                BufHandle:=Nil;
    end;          {if Error<>NoErr}
  end    {if Error=NoErr, TRUE}
  else
  begin
    {there was a memory allocation error}
    SysBeep(10);
    NbOfBytes:=Error;
  end;   {if Error=NoErr, FALSE}
end;   {SetSerBuf}

Begin
  SetSerBuf(PrinterOrModem,BufHandle,
    NbOfBytes);
End. {Main Block}
```

This routine disposes of the buffer created by SetSerialBuf and reassigns the serial port back to it's default buffer. ClearSerBuf takes two parameters, SetSerBuf(Device; BufHandle). Device specifies the printer or modem port and should be the same value that was used when SetSerBuf was called. BufHandle contains the address to the buffer generated by SetSerBuf.

```
Program Ext_ClearSerBuf;

Uses
  MemTypes,QuickDraw,OSIntf,ToolIntf,
  PackIntf;
{$D+}
{$R-}
var
  BufHandle:Handle;
  PrinterOrModem,NbOfBytes:Integer;

procedure ClearSerBuf(
  var PrinterOrModem:Integer;
  var BufHandle:Handle);

var Error,RefNum:Integer;

begin
  {test for validity of passed Handle}
  if BufHandle<>Nil then
  begin
    {set RefNum according to the port}
    if PrinterOrModem=0 then
      RefNum:=-8
    else
      RefNum:=-6;
    {Disable usage of our buffer}
  Error:=SerSetBuf(RefNum,Ptr(BufHandle^)
    ,0);
    {Test for errors}
    if Error=NoErr then
    begin
      {Unlock and Dispose of the Handle}
      HUnLock(BufHandle);
DisposHandle(BufHandle);
      BufHandle:=Nil;
    end  {if Error=NoErr, TRUE}
    else
    {Problem encountered}
      SysBeep(10);        {if Error=NoErr, FALSE}
  end    {if BufHandle<>Nil, TRUE}
  else
    {Non-Valid Handle was passed}
    SysBeep(10);
    {if BufHandle<>Nil, FALSE}
end;   {ClearSerBuf}

Begin
  ClearSerBuf(PrinterOrModem,BufHandle);
End. {Main Block}
```

External Areas

External Areas provide a way for an external procedure to take complete control of an area on a 4th Dimension input layout. You could, for example, allow the user to draw shapes using QuickDraw, implement a full-function word processor, or display a picture format that is not supported by 4th Dimension such as TIFF.

You need at least two external procedures to operate an external area. One of the external procedures will control the activity in the external area. One or more external procedures will be used to exchange information with the external area procedure. When the 4th Dimension input layout is opened, the external area procedure will be called and should create a data structure in memory. This data structure will be used by other external procedures to 'talk' with the external area. In this way your 4th Dimension procedure can know what is happening in the external area and it can provide the external area procedure with information about other user operations.

You define an external area on a 4th Dimension layout by drawing a variable field and specifying it as an External Area. The variable name will be used to store the address of the data structure defined in your external area. You enter the name of the external area procedure in the format box in the variable definition dialog. (The box may be labeled as 'Not Used'.) The size of the variable box on the layout will determine the size and location of the rectangle which your external area procedure can control.

Procedures that are to be used as external area procedures must be defined as follows:

```
procedure MyExtArea(
  var AreaEvent:EventRecord;
  var AreaRect:Rect;
  var AreaName:str255;
  var AreaHndl:MyDataHndl);
```

EventRecord, Rect, and str255 are defined in Inside Macintosh.

AreaEvent is used to inform your external area of Macintosh events.

AreaRect contains the current location of your external area on the screen.

AreaName contains the name of the 4th Dimension variable on the layout which defines the external area.

AreaHndl contains is a handle to your data structure defined as type MyDataHndl. AreaHndl is used to exchange information between your 4th Dimension procedure and your external area procedure.

You can test for the following events via the EventRecord.what parameter:

InitEvt 16	
DeInitEvt 17	
CursorEvt	18
SelectReq	20
ActivNote	21
DeActivNote	22
NotUsed	23
NotUsed	24
ScrollEvt 25	
DrawReq	26
UpdateEvt	predefined in the Toolbox
MouseDown	predefined in the Toolbox

Event 16 is informing you that your routine has been called for the first time and you should create any data structures required by your external area. (Define AreaHndl.)

Event 17 is informing you that your routine is about to be removed and you should clean-up and dispose of any data structures you created. (Dispose of AreaHndl.)

Event 18 is informing you that the cursor is over the area defined for your external area.

Event 20 is a request for you to accept all keyboard events. If you return the value 101 in the message element, you will receive all subsequent keyboard events until the user clicks on another item in the input layout.

Event 21 is sent only if you requested to be selectable when you received event 20. The receipt of event 21 means that your area has been selected and you should prepare to receive keyboard events.

Event 22 is sent only if you requested to be selectable when you received event 20. The receipt of event 22 means that your area has been deselected and you should not continue testing for keyboard events.

Event 25 is informing you that the window is being scrolled. This is so you can update the rectangle coordinates you are using. DO NOT update the screen when this event has been received. You will receive an UpdateEvt if the screen needs to be updated.

Event 26 is a request for you to display your picture in the 4th Dimension Layout Editor in place of the 1's and 0's displayed by 4th Dimension. If you return the value 102 in the message element, 4th Dimension will not draw the area occupied by the defined variable associated with the external area procedure. If you accept responsibility for drawing the area, then you must draw a picture or nothing will be displayed.

You do not directly pass parameters to an external area procedure, rather you access the data structure defined as type

MyDataHndl and place information in the data structure for the external area to recognize and act upon.

For example, the following external area procedure detects a mouse down event within an external area. When a mouse down is detected in the external area, a 1 will be stored in the data structure pointed to by AreaHndl. This external area procedure is useful for defining invisible buttons on a 4th Dimension input layout.

```
Program Ext_AreaButton;

Uses MemTypes, QuickDraw, OSIntf,
      ToolIntf, PackIntf;

Const
  InitEvt=16;
  DeInitEvt=17;

Type
  {This is the definition of the data}
  {structure I want to use.  All I want}
  {to know is if the Mouse was clicked}
  {in the external area.  0 if NO,}
  {1 if YES.}
  MyData =       record
                                      Status:Integer;
                                      end;
  MyDataPtr = ^ MyData;
  MyDataHndl = ^MyDataPtr;

Var
  AreaEv:EventRecord;
  AreaRec:Rect;
  AreaNam:str255;
  AreaHnd:MyDataHndl;

procedure AreaButton(
  var AreaEvent:EventRecord;
  var AreaRect:Rect;
  var AreaName:str255;
  var AreaHndl:MyDataHndl);

var tPoint:Point; {Tracks the mouse location.}

begin
  Case AreaEvent.What of
    InitEvt:
      begin
        {Create a new handle for our}
        {structure.}
        AreaHndl:=MyDataHndl(NewHandle(
                        sizeof(MyData)));
        {Initialize the setting to 0. Not}
        {clicked.}
        AreaHndl^^.Status:=0;
      end;       {InitEvt}
    MouseDown:
      begin
        {Wait for mouse up so we know they}
        {completed the click within our}
        {area and didn't drag out of the}
        {area before releasing.}
        While StillDown do begin end;
        {Is the mouse in our area?}
        GetMouse(tPoint);
        {AreaRect is our rect so test if the}
        {point is in our area.}
        if(PtInRect(tPoint, AreaRect)) then
                {They clicked in our area so}
                {Status=1.  Clicked}
                AreaHndl^^.Status:=1;
      end;       {MouseDown}
    DeInitEvt:
      begin      {Get rid of our handle}
                 DisposHandle(Handle(AreaHndl));
      end;       {DeInitEvt}
  end; {Case AreaEvent.What}
end; {AreaButton}

Begin
  AreaButton(AreaEv, AreaRec, AreaNam,
    AreaHnd);
End.   {Main Block}
```

The external area procedure above will track whether a mouse down has occurred in the external area. Now we need to check it's status from our 4th Dimension procedure. We check the status by calling another external procedure which accesses the data structure we created and returns the current setting of AreaHndl^^.Status. Once we've tested the variable, we set AreaHndl^^.Status back to 0, (not clicked), so we can test for the next mouse down.

We check the current setting with the following external procedure:

```
Program Ext_GetButton;

Uses MemTypes, QuickDraw, OSIntf,
      ToolIntf, PackIntf;

Type
  MyData =       record

Status:Integer;                                    end;
  MyDataPtr = ^ MyData;
  MyDataHndl = ^MyDataPtr;

Var
  Areahnd:MyDataHndl;
  Result:Integer;

procedure GetButton(
  var AreaHndl:MyDataHndl;
  var Result:Integer);

begin
  {Result returns Status}
  Result:=AreaHndl^^.Status;
  {Set Status to Not Clicked}
  AreaHndl^^.Status:=0;
end;   {GetButton}

Begin
  GetButton(AreaHnd, Result);
End.
```

From 4th Dimension we would call this procedure as GetButton(MyButton;Clicked). Where MyButton is the name of the 4th Dimension variable defining the space for the external area. (Remember, the variable name contains the address of the data structure in memory.) Clicked is a 4th Dimension global variable which will return the current setting of AreaHndl^^.Status.

When writing and installing external areas it is important to note that once you have assigned the external area to a variable in the layout editor, the External is called by the layout editor. Therefore, it is important that your routine be ready to be called immediately.

4D External Mover

There are 3 resource types used for 4th Dimension externals plus any additional resources required by your external procedure. They are 4DEX, 4DTE, and 4BND.

4DEX contains the code for your external procedure. Each procedure must have only one 4DEX resource. Therefore, if your procedure compiles to more than one code segment, you must use linker commands to join the code segments.

4DTE contains the comments about the external procedure. The information in 4DTE resources, appear in the comments field in 4D External Mover, for the associated 4DEX code.

4BND contains the bundling information needed by 4D External Mover to properly copy your external routines and bundled resources to other files. Bundling ensures that when you use the 4D External Mover to transfer an external to another file, that you get all necessary resources which have been bundled with that external procedure.

When 4D External Mover copies external procedures and any associated resources, specified in the bundle, it checks the global resource id's of resources currently in the file. If the resource id of the external you want to install is already in use, then 4D External Mover will change the global id of the resource being copied. If the id changes, 4D External Mover will also update the bundle information for that external procedure. Because the resource id may change when the resource is later copied, it is important to not access needed resources directly via the global resource id, unless you are sure the id is correct. Following is example code for obtaining the global id of resources which may have moved. This code is from the 4th Dimension Utilities and Developers Notes.

```
{The CurProcResid function is assembly}
{code which returns the 4BND resource}
{number.}

function CurProcResid:Integer;
INLINE $3EAD, $DE4E;

function GetResNum(T:ResType;  Localid:Integer):Integer;
FORWARD;

.
. Your code here
.

{Pass GetResNum the resource type and}
{the local id and it will return the}
{global id.}

function GetResNum;

type
  ResEntry = record
                    Typ:ResType;
                    LocalNum, GlobalNum:Integer;
                    end;
  ResTab = record
                    nbEntry:Integer;
                    Tab:array[1..1] of ResEntry;
                    end;
  ResTabPtr = ^ResTab;
  ResTabHandle = ^ResTabPtr;

var
  i:Integer;
  r:ResTabHandle;
  Done:Boolean;

begin
  GetResNum:=0;
  Done:=False;
  i:=1;
  {Get the pointer to 4BND resource}
  r:=POINTER(GetResource('4BND',
    CurProcResId));
  {make sure we found the resource}
  if r<> Nil then
  begin
    {Cycle thru all the entries in the}
    {4BND resource}
    while((i<=r^^.nbEntry) and
      (Not(Done))) do
      begin
        {loop until we find the one with}
        {the localID and the type we want}
        with r^^.Tab[i] do
        begin
          {check for a match}
          if (t=Typ) and
            (LocalNum=LocalId) then
          begin
            {we found it, so return the}
            {GlobalID to the calling routine}
            GetResNum:=GlobalNum;
            Done:=True;
          end;        {if}
        end; {with}
        i:=i+1;       {increment our entry counter}
    end;   {while}
  end;   {if r<>Nil}
end;   {GetResNum}
```

4th Dimension and external procedures can significantly improve your programming efficiency. 4th Dimension can fulfill most of your program development and data management needs. External procedures can supplement 4th Dimension by providing an open architecture and greater flexibility. This document provided the information you need to create and install external procedures to 4th Dimension. I hope this will help you to create externals to make your 4th Dimension databases and applications insanely great.

Programmer's Workshop

Pop-up Menus CDEF

Pascal

James Plamondon
Berkeley, CA
MacTutor Vol. 4 No. 9

Pop-up Menu Control Definition Function

by James Plamondon

[James Plamondon has a BS in Computer Science from the University of New Mexico, in addition to a BS and Master's work in Geology. He has programmed professionally on the Mac for one year, and about two years before. He is currently working at Abacus Concepts, makers of StatView.]

The Challenge

I like a challenge as much as the next Guy. Like Sir Edmund Hillary, I sometimes accept challenges just because they're there. But it's always satisfying when I can do some programming which is fun for me, and beneficial to my employer as well.

In Inside Macintosh, Volume Five, on page 242, I found my most recent Everest. That page discusses how to use the new Toolbox function PopUpMenuSelect(). Near the end of the page is the brief notation: "[using PopUpMenuSelect()] could be handled by creating a pop-up menu control within the application."

That sounded useful, so I called Mac Developer Services (on AppleLink) and asked for a copy of the pop-up menu control's definition function To my surprise, it didn't exist — or at least, they didn't have one.

There was the challenge. With a pop-up menu Control Definition Function (CDEF), pop-up menus could be used as easily as radio buttons or scroll bars. The application that I was writing for my employer at the time would also make use of pop-up menus, so I could even justify the time I spent on it! What a deal!

Why Use a CDEF?

Have you ever written code to draw a pushbutton — such as the OK or Cancel buttons in a dialog? Have you ever written code to draw a scroll bar, hilite an up-arrow, or draw a thumb region being dragged?

Probably not. All of those operations are performed by Apple's standard CDEFs. Because Apple includes these in all of its System files, you never have to worry about drawing or manipulating these controls. The CDEFs do all of the work, without your even paying attention to it.

What would happen if Apple didn't provide these standard CDEFs? Every button would look and feel different. (Did I say "look and feel?" Sounds like a lawyer!) Imagine how different Mac software would be today if all pushbuttons, radio buttons, checkboxes, and scroll bars behaved differently. (And think how different MS-Windows would be!) The standardization of such controls greatly enhanced the Mac's ease of use, both for the user and the programmer. All this, because of the much-neglected CDEF.

The pop-up menu CDEF described herein extends that standardization to pop-up menus, by making pop-up menus trivial to program, just as buttons and checkboxes are now. Just pop the CDEF into your program's resource fork (or your System file), follow a few simple guidelines, and your pop-up menus will look as good as — better than! — Apple's own.

The Language

The pop-up menu CDEF was written in MPW Pascal. It is about 4K in size, which is pretty big, for a CDEF. It would be alot smaller in Assembler; even C would probably produce a smaller CDEF. But everybody knows Pascal, and my employer uses Pascal, so it's in Pascal.

CDEFs

CDEFs are described in loving detail on pages 328-332 of Inside Mac, Volume One. They are code resources of type 'CDEF', which are loaded into memory whenever a control using the CDEF is created. Like other _DEFs (MDEFs, LDEFs, etc.), the code resource has a single entry point at its first byte (*i.e.*, no jump table). The entry point must be a function with the following definition:

```
FUNCTION MyControl(varCode: INTEGER;
    theControl: ControlHandle;
    message: INTEGER;
    param: LONGINT): LONGINT;
```

The function and argument names may be changed, of course, but their order and type must be as shown here (from IM v1 p329).

It is assumed (although not required) that there will be a separate message-handling routine for each message; all MyControl() needs to contain is a CASE statement, calling the different message-handling routines to handle the different messages. That is the organization this CDEF uses.

CDEF Messages

There are nine different messages a CDEF may receive, numbered 0 to 8. The first three, drawCntl, testCntl, and calcCRgns, must be handled by all CDEFs. When the CDEF gets a **drawCntl** message, it needs to draw the control. The 'param' argument will contain the part code of the part that needs to be redrawn, or 0 if the whole control is to be redrawn. You don't need to draw anything if the control is invisible (which you can determine by looking at the control's contrlVis field). If the control is inactive (contrlHilite = 255), then you need to draw the control differently (preferably by 'greying it out'), to tell the user

© The Definitive MacTutor, Vol. 4

that it's inactive.

The **testCntl** message is sent to find out which part of the control (if any) the given point is in. The 'given point' is passed in the 'param' LONGINT argument, in local coordinates. The part code of the control part in which the point lies should be returned by MyControl() — or return 0 if the point is not in the control. This is the only message for which the pop-up menu CDEF returns a meaningful function result; for all other messages, it just returns 0.

The next two CDEF messages, **initCntl and dispCntl**, need only be implemented by the CDEF if each control handled by the CDEF needs to be initialized in some way, and then have its initialization voided when the control is disposed of. In this CDEF, for example, each control is allocated a memory block from the heap to hold color information and a handle to its pop-up menu. This storage is deallocated when the control is disposed of.

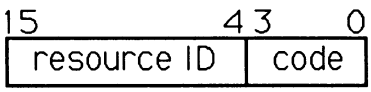
The next three CDEF messages, **posCntl, thumbCntl, and dragCntl**, need to be implemented only if your control has an indicator region (like the thumb of a scroll bar) that can be repositioned by the user. The pop-up menu CDEF has no such indicator, so when the CDEF receives these messages, it just returns without doing anything.

The last CDEF message, **autoTrack**, if this CDEF's bread and butter. The autoTrack message is sent to the CDEF whenever the user clicks inside an active control. The 'param' field will contain the part code in which the mouse lies. The pop-up menu CDEF has only one part — the pop-up menu box, part code 1 — so it doesn't even check the part code. It just calls PopUpMenuSelect(), updates the control's data to reflect the user's choice, and redraws the control with the newly-selected item in the pop-up box.

So, the CDEF does all the work. How, then, can we associate a control with this pop-up menu CDEF? By using the proper control procID, that's how.

procIDs

Which CDEF a control uses is specified by the procID field of the control specification, whether in a 'CNTL' resource or (yuck!) in a NewControl() call. The procID contains both the resource ID of the CDEF to use for the control, and also a 'variation code' specifying what sub-type of control it is. The CDEF's resource ID goes into the high three nibbles of the procID, while the variation code goes into the low nibble (a nibble is half a byte - really!):

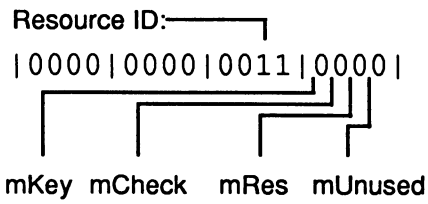


Pushbuttons, checkboxes, and radio buttons all use the CDEF with resource ID zero (0). They have variation codes 0, 1, and 2, respectively. Since the high three nibbles hold 0 (the resource ID), the procID for these controls is the same as their variation code.

Scroll bars use CDEF 1, and have variation code 0. The resource ID (1) goes into bit 4, zero goes into the low nibble, and voila! you get a procID of 16, just like in the manual (IM v1 p315).

Our pop-up menu CDEF has resource ID 3, so the procID will be $0030 + x (that's in hex; it's 48 + x in decimal), where 'x' is the variation code (each hex letter represents a nibble). The pop-up menu CDEF doesn't use variation codes, exactly; it uses variation code modifiers. This is because I wanted the user to be able to mix and match certain features of the CDEF. So, I use each of the four bits of the CDEFs variation code nibble as a flag for one of these features. The features, their bit positions, and their constants are:

Unused	uses Bit 0	mUnused = 1
Resource list	uses Bit 1	mRes = 2
Check item	uses Bit 2	mCheck = 4
Command keys	uses Bit 3	mKey = 8



With this system, assuming that popMenuProc = $30, you can specify a pop-up menu containing a list of resource names with the procID popMenuProc + mRes ($32). If you want the same pop-up menu, with the current item checked, you would use the procID popMenuProc + mRes + mCheck ($36). The standard pop-up menu control, without resource names, or checked items, would have the procID popMenuProc ($30).

Basic CDEF

According to the pop-up menu use guidelines (IM v5 p241-242), a pop-up menu should be inserted into the menu list just before PopUpMenuSelect() is called, and removed from the list right after the function returns. This is a pain in the behind, so I don't do it that way. The pop-up menu and it's sub-menus are inserted into the menu list when the control is defined, and removed when the control is disposed of. The guidelines say I can do this, if I really want to; so, I did.

mUnused

You can use this variation code modifier for whatever you want!

mRes

If you want the CDEF to find and install a list of resource names in the pop-up menu, you need to add mRes to the procID. This makes the CDEF look in the refCon field of the control, where it expects to find an OSType (such as 'FONT' or 'DRVR'). After initialization, the refCon field is no longer used by the CDEF, so you get it back to use it however you want.

mCheck

If you want the currently-selected menu item to be checked,

add mCheck to the procID. You must also set the check mark of the default menu item to the character you want to use as a check mark. If the default item is unmarked, a standard '✓' mark will be used.

mKey

mKey is reserved for future expansion. More on that later.

The Pop-Up Menu CDEF

The CDEF was written to implement standard, resource name list, checked item, and sub-list pop-up menus (and combinations thereof). If color resources are available, and the CDEF is running on a color system, the menus and their controls will be displayed in living color. At initialization, the CDEF calls SysEnvirons(), and stores the hasColorQD field. Whenever drawing is to be done, this Boolean is checked. If true, the colors of the current menu and/or menu item are retrieved (from the menu color information table) and the control is drawn using those colors. Otherwise, you get black and white (sigh).

Note that a pop-up menu control that uses sub-menus will draw the control (menu) title and pop-up box in the mTitle color of the currently selected menu (or sub-menu). You can see this in the example program in the 'Root' menu. The currently-selected item will be drawn using that item's colors, which may be different for each item in a menu. This is demonstrated in the "Thanks To" menu. Make sure that each menu title color has good contrast against its window's content color, or the title will be hard to read.

The CDEF draws the title of the pop-up menu as the control's title, left-justified in the control's boundsRect. The CDEF draws the pop-up menu box to the left of the control title. Note that all of the examples' pop-up menu titles have an extra blank space at the end of the title; this just makes the whole control look better.

The CDEF tries to make the pop-up box as wide as the menu it is controlling. If the menu is wider than the boundsRect will allow, then the pop-up box is clipped to the boundsRect. Menu items that are too long to fit in the clipped box are themselves clipped; an ellipsis is appended to such items to inform the user than text is missing. You can see this clipping by selecting 'Monty "Montana-Unit" Cole' in the "Thanks To" menu. Monty's name is the widest item in the menu, and so it defines the width of the menu. Since the contrlRect is narrower than the menu (see that, when the menu pops up, it covers part of the OK button?), it is narrower than Monty's name, too, so Monty's name must be clipped. (Sorry, Monty.)

With the CDEF, a mouse-click in the pop-up box will always be inside the default item (as the guidelines require). The user will never click in a pop-up box, just to have the pop-up menu appear to the left of the mouse — as might happen if the pop-up box were set to some "average" menu width, when the menu width might change (as with a font menu). When in doubt as to how wide to make the boundsRect, make it wider. Use ResEdit; it will call the CDEF to draw the control, and you can drag its image around. Neat!

The only way to make the left edges of pop-up boxes line up with other pop-up boxes is to leave the controls untitled, using staticText items as titles. The only way to make the right-hand edges of the pop-up boxes line up is to make the boundsRect of each of the controls no wider than that required by the narrowest control.

The Example Program

The example program is a very stripped Mac application. It simply initializes all of the usual managers, gets a dialog containing two buttons, a static text item, and three pop-up menus, and displays it. Notice that no filterProc is passed to ModalDialog(), and yet the pop-up menus get drawn, put up their menus where selected, and so on. The CDEF is doing all of the work!

After the example program exits the ModalDialog() loop, it simply disposes of the dialog and quits. In a real program, you'd get the values of the last selection from each pop-up menu control by calling GetCtlMax() to get the menuID of the menu from which the last selection was made and GetCtlMin() to get the item number of the last selection in that menu.

Using Pop-Up Menu Controls

To use a pop-up menu control in your program, you need to place the CDEF into your application resource file. You should also create two resources for each control: a 'MENU' resource, defining the menu to be popped-up (and further 'MENU' resources for its sub-menus, if any), and a 'CNTL' resource, defining the pop-up menu control. The pop-up 'MENU' resource is exactly like a regular 'MENU' resource, and its fields are all interpreted in the usual way. (See "Command-Key Equivalents" below for one possible exception: the menu title.) You can use regular 'mctb' resources to color your menus and their controls.

The pop-up menu's 'CNTL' resource has the same fields as all other 'CNTL' resources, but a number of them are interpreted and used differently. We'll go over these fields one at a time.

BoundsRect: The first field is an array of four integers, which define the rectangle bounding the control. If the control is to be used in a dialog, be sure to make the control's boundsRect agree with the item rect in the dialog item list. Otherwise, everything gets confused.

Value: The control's initial value must be set to the resource ID of the menu to be popped-up. The menu's menuID must match its resource ID.

Visible: Same as always.

Max: Initially, this field must contain the menuID of the menu containing the default item. If the pop-up menu has sub-menus, this value may be different from the pop-up menu's menuID. After the user has made a selection, this field will contain the menuID of the menu from which the item was chosen.

Min: Initially, this field must contain the item number of the default item. After the user has made a selection, this field will contain the item number of the chosen item.

ProcID: As discussed above, the procID field will contain popMenuProc plus some combination of mUnused, mRes, and mCheck (mKey will be discussed below).

RefCon: If the procID includes the mRes modifier, then this field must contain the OSType of the resource to be listed in the menu. (See 'CNTL' 129 for an example.)

Title: Surprisingly enough, this is the title of the control.

There are two other fields of importance. These are fields of the ControlRecord allocated from the 'CNTL' resource. First is the **contrlData** field, which the CDEF uses to store control-specific information. Do not mess around with this field. The other field of note, **contrlAction**, contains (-1); this means that the CDEF has a default action procedure. (See IM v1 p323-324; also, v1 p328-332.)

Command-Key Equivalents

DO NOT USE COMMAND-KEY EQUIVALENTS IN YOUR POP-UP MENUS. They will not be recognized, and will only confuse the user. I think I know how to make command-key equivalents work in pop-up menu controls, but I didn't try to implement the scheme.

I didn't make command-key equivalents work for a number of reasons. First, command-key equivalents don't really fit into the user interface, with regard to pop-up menus. Second, the implementation would require patching MenuKey() with GetTrapAddress() and SetTrapAddress(). Apple has posted signs all over these routines saying "future compatibility not guaranteed." Patching them would be like driving the family car beyond the "End of County-Maintained Road" sign. Third, it would be a lot of work.

The mKey variation code modifier has been reserved for the hardy soul who ventures to implement them. If you manage to make it fly, I'd love to hear from you. You can reach me at Abacus Concepts, 1984 Bonita Avenue, Berkeley, CA 94704, (415) 540-1949.

Miscellaneous Notes

There are a few other points that need to be mentioned before you dive into the code. First, again, the resource ID and menu ID of the pop-up menu and each of its sub-menus must match.

Second, if you use sub-menus, make sure that the default menu item is not the parent of a sub-menu. Also, be sure not to use circular references in your sub-menus.

Third, don't specify a 'cctb' resource for the pop-up menu control, because it will just get ignored. The control uses the menu color information table to color the control and its menus. Use 'mctb' resources to color your menus and their controls. Note that the pop-up control title is in essence the title of the pop-up menu, and is colored accordingly. Also note that the window's content color is used instead of the menu bar color for the title's background color.

Just before this article went to press, I added some code to allow the use of this CDEF with menu types other than the standard textMenuProc. The code requires that the MDEF defining the menu respond to two additional messages: the mItemRectMsg (512), and the mDrawItemMsg (513).

When a MDEF receives a mItemRectMsg, it should return the dimensions the given menu item in the given rectangle, with the topLeft point at (0,0) and the botRight point at (width, height).

When a MDEF receives a mDrawItemMsg, it should draw the given menu item in the given rectangle, clipped or scaled as needed.

I have not tested this code with as many custom MDEFs as I would like, but it works fine with those I've tested. It really should lock the MDEF before calling it, and restore its lock state after the MDEF returns, though. I'll save that for version 2.0 ($300 upgrade!).

Acknowledgements

A number of people contributed to this project, and I would like to thank them: Dan, Jim, and Will at Abacus, for employing me; Bryan Stearns, for his text-clipping algorithm; my wife, for putting up with my 16-hour days; Mark Williams, who let me borrow his color Mac II; and most of all, **Mark Bennet** at Apple's Mac Developer Technical Support, for his efforts above and beyond the call, even after 9pm, on a busy day, and with a nasty Spring cold. It's people like him that make Apple special. Hip-Hip, Hooray!


```
#   File:       PopMenus.make
#   Target:     PopMenus
#   Sources:    PopMenus.p PopMenus.r
#   Created:    Thursday, April 14, 1988 3:21:49 AM


PopMenus ƒƒ  PopMenus.r                 ∂
                PopMenuCDEF.r           ∂
                PopMenuCDEF.CDEF
    Rez -rd PopMenus.r -o PopMenus

PopMenus ƒƒ  PopMenus.p.o               ∂
                PopMenuCDEF.CDEF        ∂
                PopMenus.r
Link PopMenus.p.o                       ∂
    "(Libraries)"Runtime.o                  ∂
    "(PLibraries)"Paslib.o                  ∂
    -o PopMenus

PopMenuCDEF.CDEF ƒƒ PopMenuCDEF.p.o
    Link -sg PopMenuCDEF                    ∂
         -rt CDEF=1                         ∂
         -m MYCONTROL PopMenuCDEF.p.o   ∂
        "(Libraries)"Interface.o                        ∂
        "(PLibraries)"Paslib.o              ∂
        -o PopMenuCDEF.CDEF

PopMenus.p.o ƒ   PopMenus.p
    Pascal PopMenus.p -o PopMenus.p.o

PopMenuCDEF.p.o  ƒ   PopMenuCDEF.p
    Pascal PopMenuCDEF.p -o PopMenuCDEF.p.o

# ############################   #
# END OF FILE:  PopMenus.make  #
# ############################   #
```

```
{*****************************************************
Pop-up Menu Example Program
*****************************************************}

PROGRAM PopMenus;

USES MemTypes,
    Quickdraw,
```

```
    OSIntf,
    ToolIntf,
    PackIntf;

CONST
    myDLOGid =  128;  { DLOG resource ID    }

    Activate     =   2;  { Activate button }
    PopMenu1 =   3;  { popMenu control 1   }
    PopMenu2 =   4;  { popMenu control 2   }
    PopMenu3 =   5;  { popMenu control 3   }

    ON           =   0;  { for HiliteControl       }
    OFF          = 255;  { for HiliteControl       }


VAR
    dPtr:    DialogPtr;   { our test dialog }
    itemHit: INTEGER; { user's choice       }
    state:   INTEGER; { ON or OFF       }



{*****************************************************
doHitActivate: Toggles the activation state of the
    popMenu control.  Also toggles the Activate
    button's title (Activate <=> Deactivate).
*****************************************************}

PROCEDURE doHitActivate(dPtr: DialogPtr;
            VAR state: INTEGER);
VAR
    ik: INTEGER;
    ih: Handle;
    ib: Rect;

BEGIN
    { get handle to Activate button }
    GetDItem(dPtr, Activate, ik, ih, ib);

    { toggle state variable and button title }
    if (state = ON) then
        begin
            state := OFF;
            SetCTitle(ControlHandle(ih), 'Activate');
        end
    else begin
        state := ON;
        SetCTitle(ControlHandle(ih), 'Deactivate');
    end;

    { toggle the popMenu controls' activation states }
    GetDItem(dPtr, PopMenu1, ik, ih, ib);
    HiliteControl(ControlHandle(ih), state);

    GetDItem(dPtr, PopMenu2, ik, ih, ib);
    HiliteControl(ControlHandle(ih), state);

    GetDItem(dPtr, PopMenu3, ik, ih, ib);
    HiliteControl(ControlHandle(ih), state);
END;  { doHitActivate }



{*****************************************************
Main
*****************************************************}

BEGIN
    { perform the ritual incantation }
    InitGraf(@thePort);
    InitFonts;
    FlushEvents(everyEvent, 0);
    InitWindows;
    InitMenus;
    TEInit;
    InitDialogs(NIL);
    InitCursor;

    { read in the dialog from its resource template }
    dPtr := GetNewDialog(myDLOGid, NIL, POINTER(-1));

    { cycle through ModalDialog() until itemHit = OK }
    REPEAT
        ModalDialog(NIL, itemHit);

        { if the user hit the activate button, toggle }
        IF (itemHit = Activate) THEN BEGIN
            doHitActivate(dPtr, state);
        END;
    UNTIL itemHit = OK;

    DisposDialog(dPtr);
END.  { program PopMenus }

{*****************************************************
END OF FILE:  PopMenus.p
*****************************************************}

/*****************************************************
Pop-up Menu Example - Resources
*****************************************************/

#include "Types.r"
#include "PopMenuCDEF.r"

/* 1 = use color resources; 0 = don't */
#define  COLOR    1


/* include the CDEF */
data 'CDEF' (pmCDEFResID, "popMenu") {
    $$resource("PopMenuCDEF.CDEF", 'CDEF', 1)
};

/* our sample dialog */
resource 'DLOG' (128) {
    {40, 70, 170, 440},        /* 130 tall, 370 wide */
    dBoxProc,
    visible,
    noGoAway,
    0x0,
    128,
    ""
};

/* the sample dialog's item list: 3 pop-up menu ctls */
resource 'DITL' (128) {
    {   /* array DITLarray: 4 elements */
        /* [1] */
        {10, 275, 30, 360},
        Button {
            enabled,
            "OK"
        },

        /* [2] */
        {40, 275, 60, 360},
        Button {
            enabled,
            "Deactivate"
        },

        /* [3] */
        {10, 10, 30, 270},
        Control {
            enabled,
```

```
            128
        },

        /* [4] */
        {40, 10, 60, 270},
        Control {
            enabled,
            129
        },

        /* [5] */
        {70, 10, 90, 270},
        Control {
            enabled,
            130
        },

        /* [6] */
        {90, 10, 125, 360},
        StaticText {
            disabled,
            "PopUp Menu Control CDEF example.  "
            "James Plamondon,  Abacus Concepts,  "
            "(415) 540-1949."
        }
    }
};

resource 'CNTL' (128) {
    {10, 10, 30, 270},    /* rect:  contrlRect
*/
    128,              /* value: menu rsrc ID*/
    visible,          /* vis:  standard     */
    128,              /* max:  default menuID       */
    2,                /* min:  default item #       */
    popMenuProc   /* ProcID:  3         */
        + mCheck,     /* var: Check selection       */
    0,                /* rfCon: for user's use      */
    "Thanks To: "     /* title: standard            */
};

resource 'MENU' (128) {
    128,
    textMenuProc,
    allEnabled,
    enabled,
    "Thanks To: ",
    {   /* 11 items */
        "Mark Williams",
            noIcon, noKey, noMark, plain;
        "Mark Bennet",
            noIcon, noKey, appleChar, plain;
        "Joseph Daniel",
            noIcon, noKey, noMark, plain;
        "Dr. Don Morrison",
            noIcon, noKey, noMark, plain;
        "Andrew Stone",
            noIcon, noKey, noMark, plain;
        "Eleanor Plamondon",
            noIcon, noKey, noMark, plain;
        "Bruce Wampler",
            noIcon, noKey, noMark, plain;
        "Patricia Guffey",
            noIcon, noKey, noMark, plain;
        "Greta Shaw",
            noIcon, noKey, noMark, plain;
        "Monty \"Montana-Unit\" Cole",
            noIcon, noKey, noMark, plain;
        "Dr. Bernard Moret",
            noIcon, noKey, noMark, plain
    }
};

resource 'CNTL' (129) {
    {40, 10, 60, 270},    /* rect:  contrlRect
*/
    129,              /* value: rsrc ID     */
    visible,          /* vis:  standard     */
    129,              /* max: default  menuID       */
    2,                /* min:  default item #       */
    popMenuProc   /* ProcID: 3          */
        + mRes        /* var: add res names */
        + mCheck,     /* var: Check selection       */
    'FONT',           /* rfCon: OSType      */
    "Fonts: "             /* title: standard
*/
};

resource 'MENU' (129) {
    129,
    textMenuProc,
    allEnabled,
    enabled,
    "Fonts: ",
    {   /* 0 items */
    }
};

resource 'CNTL' (130) {
    {70, 10, 90, 270},    /* rect:  contrlRect
*/
    130,              /* value: rsrc ID     */
    visible,          /* vis:  standard     */
    133,              /* max:  default menuID       */
    1,                /* min: default  item #       */
    popMenuProc       /* ProcID: 3          */
        + mCheck,     /* var: Check selection       */
    0,                /* rfCon: for user's use      */
    "Root: "          /* title: standard            */
};

resource 'MENU' (130) {
    130,
    textMenuProc,
    allEnabled,
    enabled,
    "Root: ",
    {   /* 2 items */
        "Root Item1",
            noIcon, parent, "\0D131", plain;
        "Root Item2",
            noIcon, parent, "\0D132", plain
    }
};

resource 'MENU' (131) {
    131,
    textMenuProc,
    allEnabled,
    enabled,
    "",
    {   /* 2 items */
        "Sub-1 Item1",
            noIcon, noKey, noMark, plain;
        "Sub-1 Item2",
            noIcon, noKey, noMark, plain;
        "Sub-1 Item3",
            noIcon, parent, "\0D133", plain
    }
};

resource 'MENU' (132) {
    132,
    textMenuProc,
    allEnabled,
    enabled,
    "",
    {   /* 2 items */
```

```
        "Sub-2 Item1",
            noIcon, noKey, noMark, plain;
        "Sub-2 Item2",
            noIcon, noKey, noMark, plain;
        "Sub-2 Item3 (a very, very, very wide item)",
            noIcon, noKey, noMark, plain
    }
};

resource 'MENU' (133) {
    133,
    textMenuProc,
    allEnabled,
    enabled,
    "",
    {   /* 2 items */
        "Sub-3 Item1",
            noIcon, noKey, noMark, plain;
        "Sub-3 Item2",
            noIcon, noKey, noMark, plain;
        "Sub-3 Item3",
            noIcon, noKey, noMark, plain
    }
};

#if COLOR
resource 'mctb' (128) {
    {   /* array MCTBArray: 12 elements */
        /* [1] */
        128, 0,          /* menu 128, title        */
        {   /* array: 4 elements */
            /* [1] */
            0, 0, $FFFF,
            /* [2] */
            0, 0, 0,
            /* [3] */
            0, $FFFF, 0,
            /* [4] */
            $FFFF, $FFFF, $FFFF
        },

        128, 1,          /* menu 128, item 1       */
        {   /* array: 4 elements */
            /* [1] */
            $FFFF, 0, 0,
            /* [2] */
            $0040, $FFFF, 0,
            /* [3] */
            0, 0, 0,
            /* [4] */
            0, 0, 0,
        },

        128, 2,          /* menu 128, item 2       */
        {   /* array: 4 elements */
            /* [1] */
            $FFFF, 0, 0,
            /* [2] */
            $0100, $8000, 0,
            /* [3] */
            0, 0, 0,
            /* [4] */
            0, 0, 0,
        },

        128, 3,          /* menu 128, item 3       */
        {   /* array: 4 elements */
            /* [1] */
            $FFFF, 0, 0,
            /* [2] */
            $0400, $6000, 0,
            /* [3] */
            0, 0, 0,
            /* [4] */
            0, 0, 0,
        },

        128, 4,          /* menu 128, item 4       */
        {   /* array: 4 elements */
            /* [1] */
            $FFFF, 0, 0,
            /* [2] */
            $0800, $4000, 0,
            /* [3] */
            0, 0, 0,
            /* [4] */
            0, 0, 0,
        },

        128, 5,          /* menu 128, item 5       */
        {   /* array: 4 elements */
            /* [1] */
            $FFFF, 0, 0,
            /* [2] */
            $1000, $2000, 0,
            /* [3] */
            0, 0, 0,
            /* [4] */
            0, 0, 0,
        },

        128, 6,          /* menu 128, item 6       */
        {   /* array: 4 elements */
            /* [1] */
            $FFFF, 0, 0,
            /* [2] */
            $1800, $1800, 0,
            /* [3] */
            0, 0, 0,
            /* [4] */
            0, 0, 0,
        },

        128, 7,          /* menu 128, item 7       */
        {   /* array: 4 elements */
            /* [1] */
            $FFFF, 0, 0,
            /* [2] */
            $2000, $1000, 0,
            /* [3] */
            0, 0, 0,
            /* [4] */
            0, 0, 0,
        },

        128, 8,          /* menu 128, item 8       */
        {   /* array: 4 elements */
            /* [1] */
            $FFFF, 0, 0,
            /* [2] */
            $4000, $0400, 0,
            /* [3] */
            0, 0, 0,
            /* [4] */
            0, 0, 0,
        },

        128, 9,          /* menu 128, item 9       */
        {   /* array: 4 elements */
            /* [1] */
            $FFFF, 0, 0,
            /* [2] */
            $6000, $0200, 0,
            /* [3] */
            0, 0, 0,
            /* [4] */
            0, 0, 0,
        },
```


© The Definitive MacTutor, Vol. 4

```
        128, 10,            /* menu 128, item 10 */
        (   /* array: 4 elements */
            /* [1] */
            $FFFF, 0, 0,
            /* [2] */
            $8000, $0100, 0,
            /* [3] */
            0, 0, 0,
            /* [4] */
            0, 0, 0,
        },

        128, 11,            /* menu 128, item 11 */
        (   /* array: 4 elements */
            /* [1] */
            $FFFF, 0, 0,
            /* [2] */
            $FFFF, $0040, 0,
            /* [3] */
            0, 0, 0,
            /* [4] */
            0, 0, 0,
        )
    }
};

resource 'mctb' (129) {
    (   /* array MCTBArray: 1 elements */
        /* [1] */
        129, 0,
        (   /* array: 4 elements */
            /* [1] */
            0, $4000, $4000,
            /* [2] */
            0, 0, 0,
            /* [3] */
            0, $4000, 0,
            /* [4] */
            $FFFF, $FFFF, $FFFF
        )
    }
};

resource 'mctb' (130) {
    (   /* array MCTBArray: 1 elements */
        /* [1] */
        130, 0,
        (   /* array: 4 elements */
            /* [1] */
            $0, $4000, $0,
            /* [2] */
            0, 0, 0,
            /* [3] */
            0, $4000, 0,
            /* [4] */
            $4000, 0, $4000
        )
    }
};

resource 'mctb' (131) {
    (   /* array MCTBArray: 1 elements */
        /* [1] */
        131, 0,
        (   /* array: 4 elements */
            /* [1] */
            $8000, $8000, $2000,
            /* [2] */
            $0, $0, $0,
            /* [3] */
            $0, $0, $0,
            /* [4] */
            $2000, $2000, $8000
        )
    }
};

resource 'mctb' (132) {
    (   /* array MCTBArray: 1 elements */
        /* [1] */
        132, 0,
        (   /* array: 4 elements */
            /* [1] */
            $1000, $C000, $C000,
            /* [2] */
            0, 0, 0,
            /* [3] */
            $4000, $6000, $FFFF,
            /* [4] */
            $C000, $1000, $1000
        )
    }
};

resource 'mctb' (133) {
    (   /* array MCTBArray: 1 elements */
        /* [1] */
        133, 0,
        (   /* array: 4 elements */
            /* [1] */
            $8000, $0, $0,
            /* [2] */
            0, 0, 0,
            /* [3] */
            $4000, 0, $4000,
            /* [4] */
            $0, $8000, $0
        )
    }
};

resource 'dctb' (128) {
    0x0,
    0,
    (   /* array ColorSpec: 3 elements */
        /* [1] */
        wContentColor, $FFFF, $FFFF, $FFFF,
        /* [2] */
        wFrameColor, 0, 0, $FFFF,
        /* [3] */
        wHiliteColor, $2000, $FFFF, $2000
    )
};

data 'ictb' (128) {
    /* (000) 0000: */ $"0040 0018"
    /* (004) 0004: */ $"0040 0018"
    /* (008) 0008: */ $"0000 0000"
    /* (012) 000C: */ $"0000 0000"
    /* (016) 0010: */ $"0000 0000"
    /* (020) 0014: */ $"800F 0040"

    /* (024) 0018: */ $"0000 0000"
    /* (028) 001C: */ $"0000"
    /* (030) 001E: */ $"0003"
    /* (032) 0020: */ $"0000 0000 0000 0000"
    /* (040) 0028: */ $"0001 4000 FFFF 4000"
    /* (048) 0030: */ $"0002 0000 0000 0000"
    /* (056) 0038: */ $"0003 0000 0000 FFFF"

    /* (064) 0040: */ $"0054"
    /* (066) 0042: */ $"0200"
    /* (068) 0044: */ $"0009"
    /* (070) 0046: */ $"FFFF 0000 0000"
    /* (076) 004C: */ $"FFFF FFFF FFFF"
    /* (082) 0052: */ $"0001"
```

```
    /* (084) 0054: */ $"0647 656E 6576 61"
};
#endif   /* COLOR */

/* end of resource file */


{*****************************************************
PopMenuIntf.p
   This file contains the Pascal interface for the
   constants used in PopUpCDEF.p, and in any other
   program or unit which uses pop-ups.
*****************************************************}

UNIT PopMenuIntf;

INTERFACE
CONST
    { VARIATION CODE MODIFIERS:    }
    mUnused          =   1;
    mRes         =   2;
    mCheck           =   4;
    mKey         =   8;

    { part codes }
    inPopUpBox       =    1;
    titlePart            =    2;

    { MDEF message: get item dimensions }
    mItemRectMsg =   512;

    { MDEF message: draw item in rect }
    mDrawItemMsg =   513;
IMPLEMENTATION
END.  { PopMenuIntf }


{*****************************************************
PopMenuCDEF.p
   This file contains the Pascal source code for the
   routines needed to implement the pop-up menu
   button CDEF described (in passing) in Inside Mac,
   v5, p242.
*****************************************************}

UNIT PopMenuCDEF;

INTERFACE
USES
    {$U MemTypes.p    }   MemTypes,
    {$U QuickDraw.p   }   QuickDraw,
    {$U OSIntf.p      }   OSIntf,
    {$U ToolIntf.p        }   ToolIntf,
    {$U PopMenuIntf.p }   PopMenuIntf;


FUNCTION MyControl(varCode: INTEGER;
                   theCntl: ControlHandle;
                   message: INTEGER;
                   param: LONGINT): LONGINT;


IMPLEMENTATION
CONST
    VISIBLE      =   255;
    INVISIBLE    =     0;
    INACTIVE =   255;
    ACTIVE       =     0;
    DRAW_ALL =     0;
    NOT_IN_CTL   =     0;
    L_PIXELS =    13;
    GREY     =    16;
    PARENT       =   $1B;


TYPE
    CtlDataRec   = record
        popMenu: MenuHandle;
        menuProcID:   INTEGER;
        hasColorQD:   Boolean;
        markChar:     Char;
        wFgColor:     RGBColor;
        wBgColor:     RGBColor;
        wContColor:   RGBColor;
        mTitleColor:  RGBColor;
        mBgColor:     RGBColor;
        iNameColor:   RGBColor;
        iKeyColor:    RGBColor;
    end;
    CtlDataPtr   =   ^CtlDataRec;
    CtlDataHdl   =   ^CtlDataPtr;

    StateRec = record
        savePort:     GrafPtr;
        savePen: PenState;
        oldClip:      RgnHandle;
        newClip:      RgnHandle;
    end;


{*****************************************************
forward declarations
*****************************************************}

PROCEDURE    doDrawCntl(theCntl: ControlHandle;
                 vcLong, param: LONGINT);
                 forward;
FUNCTION     doTestCntl(theCntl: ControlHandle;
                 param: LONGINT): LONGINT;
                 forward;
PROCEDURE    doCalcCRgns(theCntl: ControlHandle;
                 param: LONGINT);
                 forward;
PROCEDURE    doInitCntl(theCntl: ControlHandle;
                 vcLong: LONGINT);
                 forward;
PROCEDURE    doDispCntl(theCntl: ControlHandle;
                 vcLong: LONGINT);
                 forward;
PROCEDURE    doAutoTrack(theCntl: ControlHandle;
                 vcLong, param: LONGINT);
                 forward;


{*****************************************************
MyControl: Main entry point.  Call appropriate
    message-handling routine.
*****************************************************}

FUNCTION MyControl(varCode: INTEGER;
             theCntl: ControlHandle;
             message: INTEGER;
             param: LONGINT): LONGINT;
VAR
    vcLong:      LONGINT;

BEGIN
    MyControl := 0;
    vcLong := Ord4(varCode);

    CASE message OF
        drawCntl:
            doDrawCntl(theCntl, vcLong, param);
        testCntl:
            MyControl := doTestCntl(theCntl, param);
        calcCRgns:
            doCalcCRgns(theCntl, param);
        initCntl:
```

```
            doInitCntl(theCntl, vcLong);
        dispCntl:
            doDispCntl(theCntl, vcLong);
        autoTrack:
            doAutoTrack(theCntl, vcLong, param);
    END;  { case }
END;  { MyControl }



{*****************************************************
CallMDEF: Calls the given ProcPtr, passing it the
    given parameters.
*****************************************************}

PROCEDURE CallMDEF(message: INTEGER;
                   theMenu: MenuHandle;
                   menuRect: Rect;
                   hitPt: Point;
                   whichItem: INTEGER;
                   MDEFProc: ProcPtr);
Inline
    $205F,      { move.l (sp)+, a0; get address of proc}
    $4E90;      { jsr (a0) ; call the proc
}



{*****************************************************
GetItemRect: Get the given item's rectangle.
*****************************************************}

PROCEDURE GetItemRect(theCntl: ControlHandle;
                              menuID: INTEGER;
                              menuItem: INTEGER;
                              VAR boxRect: Rect);
VAR
    hitPt:          Point;
    menuHdl:     MenuHandle;
    mDefProc:       Handle;

BEGIN
    SetPt(hitPt, 0, 0);
    menuHdl := GetMHandle(menuID);
    mDefProc := menuHdl^^.menuProc;
    LoadResource(mDefProc);

    CallMDEF(mItemRectMsg,
            menuHdl,
            boxRect,
            hitPt,
            menuItem,
            ProcPtr(mDefProc^));
END;  { GetItemRect }


{*****************************************************
DrawMenuItem: Draw the given menu item in the
    given rectangle.
*****************************************************}

PROCEDURE DrawMenuItem(theCntl: ControlHandle;
                              menuID: INTEGER;
                              menuItem: INTEGER;
                              boxRect: Rect);
VAR
    hitPt:          Point;
    menuHdl:        MenuHandle;
    mDefProc:       Handle;

BEGIN
    SetPt(hitPt, 0, 0);
    menuHdl := GetMHandle(menuID);
    mDefProc := menuHdl^^.menuProc;
    LoadResource(mDefProc);

    CallMDEF(mDrawItemMsg,
            menuHdl,
            boxRect,
            hitPt,
            menuItem,
            ProcPtr(mDefProc^));
END;  { DrawMenuItem }


{*****************************************************
GetContentColor: Get the window's content color.
*****************************************************}

PROCEDURE GetContentColor(wPtr: WindowPtr;
            VAR contColor: RGBColor);
VAR
    auxWinHdl:   AuxWinHndl;
    winCTable:   WCTabHandle;
    b_ignore:        Boolean;
    i:           INTEGER;

BEGIN
    b_ignore := GetAuxWin(wPtr, auxWinHdl);
    winCTable := WCTabHandle(auxWinHdl^^.
                                awCTable);

    i := winCTable^^.ctSize;

    { search for wContentColor }
    while ((i >= 0) and (winCTable^^.ctTable[i].value
                        <> wContentColor)) do begin
        i := i - 1;
    end;

    { if we didn't find it, default to first entry }
    if (i < 0) then
        i := 0;

    contColor := winCTable^^.ctTable[i].rgb;
END;  { GetContentColor }


{*****************************************************
GetMenuColors: Initialize the control's menu color
    information.  ctlData must be locked before calling
    this routine.
*****************************************************}

PROCEDURE GetMenuColors(theCntl: ControlHandle;
            ctlData: CtlDataHdl);
VAR
    WhiteRGB:    RGBColor;
    BlackRGB:    RGBColor;
    mbarPtr:     MCEntryPtr;
    titlePtr:        MCEntryPtr;
    itemPtr:     MCEntryPtr;

BEGIN
    { default colors }
    WhiteRGB.red   := $FFFF;
    WhiteRGB.green := $FFFF;
    WhiteRGB.blue  := $FFFF;
    BlackRGB.red   := 0;
    BlackRGB.green := 0;
    BlackRGB.blue  := 0;

    with theCntl^^ do begin
        mbarPtr  := GetMCEntry(0, 0);
        titlePtr := GetMCEntry(contrlMax, 0);
        itemPtr  := GetMCEntry(contrlMax, contrlMin);
    end;
```

```
    { get defaults from mbar, or default to B&W }
    with ctlData^^ do begin
        if (mbarPtr = NIL) then
            begin
                if (titlePtr = NIL) then begin
                    mTitleColor := BlackRGB;
                    mBgColor    := WhiteRGB;
                end;

                if (itemPtr = NIL) then begin
                    iNameColor  := BlackRGB;
                    iKeyColor   := BlackRGB;
                end;
            end
        else if (titlePtr = NIL) then begin
            mTitleColor := mbarPtr^.mctRGB1;
            mBgColor := mbarPtr^.mctRGB2;

            if (itemPtr = NIL) then begin
                iNameColor  := mbarPtr^.mctRGB3;
                iKeyColor   := mbarPtr^.mctRGB3;
            end;
        end;

        { get colors and defaults from the title entry }
        if (titlePtr <> NIL) then begin
            mTitleColor := titlePtr^.mctRGB1;
            mBgColor := titlePtr^.mctRGB4;

            if (itemPtr = NIL) then begin
                iNameColor := titlePtr^.mctRGB3;
                iKeyColor  := titlePtr^.mctRGB3;
            end;
        end;

        { set the item colors }
        if (itemPtr <> NIL) then begin
            iNameColor := itemPtr^.mctRGB2;
            iKeyColor  := itemPtr^.mctRGB3;
        end;
    end;  { with ctlData^^ }
END;  { GetMenuColors }


{*****************************************************
InitColorInfo: Initialize the control's color information.
*****************************************************}

PROCEDURE InitColorInfo(theCntl: ControlHandle;
            ctlData: CtlDataHdl);
VAR
    i:          INTEGER;
    wPtr:           WindowPtr;

BEGIN
    HLock(Handle(ctlData));

    with ctlData^^ do begin
        wPtr := theCntl^^.contrlOwner;

        { get the window's content color }
        GetContentColor(wPtr, wContColor);

        { save the window's current fg and bg colors }
        GetForeColor(wFgColor);
        GetBackColor(wBgColor);

        { get the menu's and current item's colors }
        GetMenuColors(theCntl, ctlData);
    end;

    HUnlock(Handle(ctlData));
END;  { InitColorInfo }


{*****************************************************
GetTitleRect: Get the title of the pop-up menu.
*****************************************************}

PROCEDURE GetTitleRect(theCntl: ControlHandle;
            VAR titleRect: Rect);
VAR
    fInfo:   FontInfo;
    height:  INTEGER;

BEGIN
    GetFontInfo(fInfo);

    with fInfo do begin
        height := ascent + descent + leading;
    end;

    { define the title's rect }
    with theCntl^^ do begin
        SetRect(titleRect, contrlRect.left,
                           contrlRect.top,
                           contrlRect.left +
                             StringWidth(contrlTitle),
                           contrlRect.top + height);

        with titleRect do begin
            if (bottom > contrlRect.bottom - 1) then
                bottom := contrlRect.bottom - 1;

            if (right > contrlRect.right - 1) then
                right := contrlRect.right - 1;
        end;  { with titleRect }
    end;  { with theCntl^^ }
END;  { GetTitleRect }


{*****************************************************
GetBoxRect: Get the box surrounding the pop-up
    box.
*****************************************************}

PROCEDURE GetBoxRect(theCntl: ControlHandle;
            VAR boxRect: Rect);
VAR
    leftEdge:           INTEGER;
    popMenu:      MenuHandle;
    fInfo:            FontInfo;
    height:           INTEGER;
    menuProcID:       INTEGER;
    ctlData:          CtlDataHdl;

BEGIN
    ctlData := CtlDataHdl(theCntl^^.contrlData);
    menuProcID := ctlData^^.menuProcID;

    if (menuProcID = textMenuProc) then
        begin
            GetFontInfo(fInfo);

            with fInfo do begin
                height := ascent + descent + leading;
            end;

            with theCntl^^ do begin
                { find the left edge of the pop-up box }
                leftEdge := contrlRect.left +
                            StringWidth(contrlTitle);

                popMenu := ctlData^^.popMenu;

                { defend against Menu Manager bug }
                CalcMenuSize(popMenu);
```


© The Definitive MacTutor, Vol. 4

```
                { define the pop-up box's rect }
                SetRect(boxRect,
                        leftEdge,
                        ctrlRect.top,
                        leftEdge +
                            popMenu^^.menuWidth +
                            2,
                        ctrlRect.top + height + 1);
            end;  { with theCntl^^ }
        end  { menuProc = nil }
    else begin
        GetItemRect(theCntl,
                    theCntl^^.ctrlMax,
                    theCntl^^.ctrlMin,
                    boxRect);
    end;  { else }

    with theCntl^^ do begin
        with boxRect do begin
            if (bottom > ctrlRect.bottom - 1) then
                bottom := ctrlRect.bottom - 1;

            if (right > ctrlRect.right - 1) then
                right := ctrlRect.right - 1;
        end;  { with boxRect }
    end;  { with theCntl^^ }
END;  { GetBoxRect }


(*****************************************************
GetCtlRect: Get the box surrounding the pop-up box
    and its title.
*****************************************************)

PROCEDURE GetCtlRect(theCntl: ControlHandle;
            VAR ctlRect: Rect);
VAR
    boxRect: Rect;
    titleRect:   Rect;

BEGIN
    GetBoxRect(theCntl, boxRect);
    GetTitleRect(theCntl, titleRect);

    UnionRect(boxRect, titleRect, ctlRect);

    with ctlRect do begin
        SetRect(ctlRect, left, top,
            right + 1, bottom + 1);
    end;
END;  { GetCtlRect }


(*****************************************************
InstallMenus: Recursive routine to install a menu and
    its sub-menus, if any.  It is only called once
    (from doInitCntl()).
*****************************************************)

PROCEDURE InstallMenus(rsrcID: INTEGER);
VAR
    mh:      MenuHandle;
    ni:      INTEGER;
    i:       INTEGER;
    c:       Char;

BEGIN
    mh := GetMenu(rsrcID);
    InsertMenu(mh, -1);
    ni := CountMItems(mh);

    { look for parent items }
    for i := 1 to ni do begin
        GetItemCmd(mh, i, c);

        { if it's a parent item, recurse on its child }
        if (c = CHR(PARENT)) then begin
            GetItemMark(mh, i, c);
            InstallMenus(ORD(c));
        end;
    end;
END;  { InstallMenus }


(*****************************************************
RemoveMenus: Recursive routine to remove a menu
    and its sub-menus, if any.  It is only called once
    (from doDispCntl()).
*****************************************************)

PROCEDURE RemoveMenus(menuID: INTEGER);
VAR
    mh:      MenuHandle;
    ni:      INTEGER;
    i:       INTEGER;
    c:       Char;

BEGIN
    mh := GetMHandle(menuID);
    ni := CountMItems(mh);

    { look for parent items }
    for i := 1 to ni do begin
        GetItemCmd(mh, i, c);

        { if it's a parent item, recurse on its child }
        if (c = CHR(PARENT)) then begin
            GetItemMark(mh, i, c);
            RemoveMenus(ORD(c));
        end;
    end;

    { delete the menu from the menu list }
    DeleteMenu(menuID);
    ReleaseResource(Handle(mh));
END;  { RemoveMenus }


(*****************************************************
ShrinkString: Make the given string fit in the given
    box.  From a program by Bryan Stearns.
*****************************************************)

PROCEDURE ShrinkString(VAR s: Str255; r: Rect);
VAR
    s_pix:   INTEGER;
    s_len:   INTEGER;
    room:    INTEGER;

BEGIN
    { how much room do we have? }
    room := (r.right - r.left) - L_PIXELS;

    { watch for weirdness }
    if (room < 0) then begin
        room := 0;
        s[0] := CHR(0);
    end;

    { get the width of the string }
    s_pix := StringWidth(s);

    { will it fit? }
    if (s_pix > room) then begin
        s_len := LENGTH(s);
        room := room - CharWidth('…');
```

```
        repeat
            s_pix := s_pix - CharWidth(s[s_len]);
            s_len := s_len - 1;
        until ((s_pix < room) or (LENGTH(s) = 0));

        s_len := s_len + 1;
        s[s_len] := '…';
        s[0] := CHR(s_len);
    end;
END;  { ShrinkString }



(*****************************************************
DrawTitle: Draw the title of the pop-up menu control.
*****************************************************)

PROCEDURE DrawTitle(theCntl: ControlHandle);
VAR
    titleRect:   Rect;
    ctlData: CtlDataHdl;
    fInfo:   FontInfo;
    baseline:    INTEGER;

BEGIN
    with theCntl^^ do begin
        ctlData := CtlDataHdl(contrlData);

        { if we need to draw in color, set the colors }
        with ctlData^^ do begin
            if (hasColorQD) then begin
                if (contrlHilite = titlePart) then
                    begin
                        RGBForeColor(wContColor);
                        RGBBackColor(mTitleColor);
                    end
                else begin
                    RGBForeColor(mTitleColor);
                    RGBBackColor(wContColor);
                end;
            end;
        end;

        { get the control's title box, and erase it }
        GetTitleRect(theCntl, titleRect);
        EraseRect(titleRect);

        { get info about the current font }
        GetFontInfo(fInfo);

        { define baseline }
        with fInfo do begin
            baseline := contrlRect.top + ascent;
        end;

        { move to baseline }
        MoveTo(titleRect.left + 1, baseline);

        { draw control title (= the pop-up menu's title) }
        DrawString(contrlTitle);

        { if we drew in color, restore the colors }
        with ctlData^^ do begin
            if (hasColorQD) then
                begin
                    RGBForeColor(wFgColor);
                    RGBBackColor(wBgColor);
                end
            else if (contrlHilite = titlePart) then begin
                InvertRect(titleRect);
            end;
        end;
    end;
END;  { DrawTitle }



(*****************************************************
DrawDropShadow: Draw the shadow around the
    pop-up box of the pop-up menu control.
*****************************************************)

PROCEDURE DrawDropShadow(
                        theCntl: ControlHandle;
                        boxRect: Rect);
VAR
    ctlData: CtlDataHdl;

BEGIN
    ctlData := CtlDataHdl(theCntl^^.contrlData);

    { if we need to draw in color, set the colors }
    with ctlData^^ do begin
        if (hasColorQD) then begin
            RGBForeColor(mTitleColor);
            RGBBackColor(mBgColor);
        end;  { if }
    end; { with ctlData^^ }

    with boxRect do begin
        { draw the drop shadow }
        MoveTo(right, top + 2);
        LineTo(right, bottom);
        LineTo(left + 2, bottom);
    end;  { with boxRect }

    { if we drew in color, restore the colors }
    with ctlData^^ do begin
        if (hasColorQD) then begin
            RGBForeColor(wFgColor);
            RGBBackColor(wBgColor);
        end;  { if }
    end;  { with ctlData^^ }
END;  { DrawDropShadow }

(*****************************************************
DrawPopBox: Draw the pop-up box of the pop-up
    menu control.  Also draws drop shadow.
*****************************************************)

PROCEDURE DrawPopBox(theCntl: ControlHandle;
                     vcLong: LONGINT);
VAR
    boxRect:     Rect;
    itemStr:     Str255;
    ctlData:     CtlDataHdl;
    fInfo:       FontInfo;
    baseline:        INTEGER;
    menuProcID:  INTEGER;

BEGIN
    ctlData := CtlDataHdl(theCntl^^.contrlData);
    menuProcID := ctlData^^.menuProcID;

    if (menuProcID = textMenuProc) then
        begin { standard textMenuProc }
            with theCntl^^ do begin
                ctlData := CtlDataHdl(contrlData);
                GetBoxRect(theCntl, boxRect);

                { erase the box and shadow }
                with boxRect do begin
                    SetPt(botRight, right + 2,
                                bottom + 2);
                    EraseRect(boxRect);
                    SetPt(botRight, right - 2,
                                bottom - 2);
```

```
            end;  { with }

            { get current selection string }
            GetItem(GetMHandle(cntrlMax),
                        cntrlMin,
                        itemStr);

            { make the string fit in the boxRect }
            ShrinkString(itemStr, boxRect);

            { if color, set the colors }
            with ctlData^^ do begin
                if (hasColorQD) then begin
                  RGBForeColor(mTitleColor);
                  RGBBackColor(mBgColor);
                end;
            end;

            { frame the box }
            FrameRect(boxRect);

            { get info about the current font }
            GetFontInfo(fInfo);

            { define baseline }
            with fInfo do begin
                baseline := cntrlRect.top +
                            ascent;
            end;

            { if color, set the colors }
            with ctlData^^ do begin
                if (hasColorQD) then begin
                  RGBForeColor(iNameColor);
                end;
            end;

            with boxRect do begin
                { draw the string in the popup box }
                MoveTo(left+L_PIXELS, baseline);
                DrawString(itemStr);
            end;  { with boxRect }

            { if color, restore the colors }
            with ctlData^^ do begin
                if (hasColorQD) then begin
                    RGBForeColor(wFgColor);
                    RGBBackColor(wBgColor);
                end;
            end;
        end;  { with theCntl^^ }
    end
  else begin { non-standard menuProc }
    GetBoxRect(theCntl, boxRect);
    DrawMenuItem(theCntl,
                 theCntl^^.cntrlMax,
                 theCntl^^.cntrlMin,
                 boxRect);
  end;

  DrawDropShadow(theCntl, boxRect);
END;  { DrawPopBox }



{*****************************************************
DrawDisabled: Invert the pop-up menu control's title.
*****************************************************}

PROCEDURE DrawDisabled(theCntl: ControlHandle);
VAR
    greyPat: PatHandle;
    ctlRect: Rect;

BEGIN
    { get the grey pattern from the System file }
    greyPat:=PatHandle(GetResource('PAT ',GREY));
    PenPat(greyPat^^);
    ReleaseResource(Handle(greyPat));

    { set the pen mode }
    PenMode(patBic);

    GetCtlRect(theCntl, ctlRect);
    PaintRect(ctlRect);
END;  { DrawDisabled }

{*****************************************************
SaveState: Save the current drawing environment.
*****************************************************}
PROCEDURE SaveState(theCntl: ControlHandle;
            VAR theState: StateRec);
VAR
    ctlData: CtlDataHdl;

BEGIN
    { lock the control handle }
    HLock(Handle(theCntl));

    with theCntl^^ do begin
        with theState do begin
            { save current grafPort; set to owner }
            GetPort(savePort);
            SetPort(cntrlOwner);
        { allocate space for clipping regions }
            oldClip := NewRgn;
            newClip := NewRgn;
            { save old clipping region }
            GetClip(oldClip);
            { set newClip region to given rectangle }
            RectRgn(newClip, cntrlRect);
            { newClip: intersection of rect and region }
            SectRgn(oldClip, newClip, newClip);
            { set grafPorts' clip region to the result  }
            SetClip(newClip);
            { save current pen state; normalize pen }
            GetPenState(savePen);
            PenNormal;
            { if we have color, get the menu color info }
            ctlData := CtlDataHdl(cntrlData);
            if (ctlData^^.hasColorQD) then begin
                HLock(Handle(ctlData));
                  GetMenuColors(theCntl, ctlData);
                HUnlock(Handle(ctlData));
            end;
        end;  { with theState }
    end;  { with theCntl^^ }

    { unlock the control handle }
    HUnlock(Handle(theCntl));
END;  { SaveState }

{*****************************************************
RestoreState: Restore the saved drawing environment.
*****************************************************}
PROCEDURE RestoreState(theCntl: ControlHandle;
            VAR theState: StateRec);
BEGIN
    with theState do begin
        { restore saved states }
        SetClip(oldClip);
        SetPenState(savePen);
        SetPort(savePort);

        { dispose of regions }
        DisposeRgn(oldClip);
        DisposeRgn(newClip);
    end;  { with }
```

```
END;  { RestoreState }

{*****************************************************
doDrawCntl: Draw the pop-up menu box and title.
*****************************************************}

PROCEDURE doDrawCntl(theCntl: ControlHandle;
            vcLong, param: LONGINT);
VAR
    theState:    StateRec;

BEGIN
    if (theCntl^^.contrlVis = VISIBLE) then begin
        { save the current drawing environment }
        SaveState(theCntl, theState);

        { lock the control }
        HLock(Handle(theCntl));

        { draw the control }
        DrawTitle(theCntl);
        DrawPopBox(theCntl, vcLong);

        { if inactive, grey out the control }
        if (theCntl^^.contrlHilite=INACTIVE) then begin
            DrawDisabled(theCntl);
        end;

        { unlock the control }
        HUnlock(Handle(theCntl));

        { restore the saved drawing environment }
        RestoreState(theCntl, theState);
    end;  { if VISIBLE }
END;  { doDrawCntl }

{*************************************
doTestCntl: Determine in which part of the control (if
    any) the given point (in 'param') lies.
*****************************************************}

FUNCTION doTestCntl(theCntl: ControlHandle;
            param: LONGINT): LONGINT;
VAR
    boxRect: Rect;

BEGIN
    if (theCntl^^.contrlHilite <> INACTIVE) then
        begin  { control is active }
            GetBoxRect(theCntl, boxRect);

            if PtInRect(Point(param), boxRect) then
                doTestCntl := inPopUpBox
            else
                doTestCntl := NOT_IN_CTL;
        end
    else  { control is inactive }
        doTestCntl := NOT_IN_CTL;
END;  { doTestCntl }

{*****************************************************
doCalcCRgns: Calculate the region the control
    occupies in its window.
*****************************************************}

PROCEDURE doCalcCRgns(theCntl: ControlHandle;
            param: LONGINT);
VAR
    boxRect: Rect;

BEGIN
    if (BitAnd(param, $80000000) = $80000000) then
        begin { wants indicator region - we have none }
            param := BitAnd(param, $0FFFFFFF);
            SetEmptyRgn(RgnHandle(param));
        end
    else begin
        param := BitAnd(param, $0FFFFFFF);

        { set the given region to boxRect }
        GetBoxRect(theCntl, boxRect);
        RectRgn(RgnHandle(param), boxRect);
    end;
END;  { doCalcCRgns }

{*****************************************************
doInitCntl: Do any initialization required for the given
    control.
*****************************************************}

PROCEDURE doInitCntl(theCntl: ControlHandle;
            vcLong: LONGINT);
VAR
    popMenu: MenuHandle;
    dfltMenu:        MenuHandle;
    ctlRect:     Rect;
    ctlData:     CtlDataHdl;
    world:       SysEnvRec;
    error:       OSErr;
    markChar:    Char;
    menuProcID:  INTEGER;

BEGIN
    { lock the control record down  }
    HLock(Handle(theCntl));

    with theCntl^^ do begin
        { allocate a relocatable block }
        ctlData := CtlDataHdl(NewHandle(sizeof(
                                    CtlDataRec)));

        { is color QuickDraw running? }
        error := SysEnvirons(1, world);
        ctlData^^.hasColorQD := world.hasColorQD;

        { store a handle to the control data  }
        contrlData := Handle(ctlData);

        { erase the control's rectangle }
        EraseRect(contrlRect);

        { get a handle to the 'MENU' resource }
        popMenu := MenuHandle(
                GetResource('MENU',contrlValue));

        { save the menuProc ID }
        ctlData^^.menuProcID := HiWord(
                        Ord4(popMenu^^.menuProc));

        { load pop-up menu, and its sub-menus }
        InstallMenus(contrlValue);
        popMenu := GetMHandle(contrlValue);

        { save the pop-up menu's menu handle }
        ctlData^^.popMenu := popMenu;

        { append resource names to the menu? }
        if ((BitAnd(vcLong, mRes) = mRes) and
                (contrlRfCon <> 0)) then begin
            AddResMenu(popMenu,
                        OSType(contrlRfCon));
        end;
        { does the user want to use a check mark? }
        if (BitAnd(vcLong,mCheck)=mCheck) then
                begin
            { get a handle to the default menu }
            dfltMenu := GetMHandle(contrlMax);
            { get the default menu item's mark char }
```

```
            GetItemMark(dfltMenu,
                        ctrlMin,
                        markChar);
            { if no mark char, default to checkMark }
            if (markChar = CHR(noMark)) then begin
                markChar := CHR(checkMark);
                { set the default item's mark }
                SetItemMark(dfltMenu,
                            ctrlMin,
                            markChar);
            end;
            { save the default item }
            ctlData^^.markChar := markChar;
        end;
        { if we have color, initialize the color info }
        if (world.hasColorQD) then begin
            InitColorInfo(theCntl, ctlData);
        end;
    { flag the default action proc }
        ctrlAction := POINTER(-1);
    end;  { with theCntl }

    { unlock the control record before SetCTitle }
    HUnlock(Handle(theCntl));
END;  { doInitCntl }

{*******************************************************
doDispCntl: Do any de-allocation required for the
    given control.
*******************************************************}
PROCEDURE doDispCntl(theCntl: ControlHandle;
            vcLong: LONGINT);
VAR
    popMenu: MenuHandle;
    ctlData:     CtlDataHdl;
BEGIN
    ctlData := CtlDataHdl(theCntl^^.ctrlData);
    popMenu := ctlData^^.popMenu;
    { remove the pop-up and its sub-menus }
    RemoveMenus(popMenu^^.menuID);
END;  { doDispCntl }

{*******************************************************
doAutoTrack: This is the default action procedure for
    all controls of this type.  TrackControl() will
    place the value inPopBox in ctrlHilite
    before calling doAutoTrack, so the old
    value will be lost before we can save it here.
*******************************************************}
PROCEDURE doAutoTrack(theCntl: ControlHandle;
            vcLong, param: LONGINT);
VAR
    popMenu:     MenuHandle;
    menuResult:      LONGINT;
    menuID:          INTEGER;
    menuItem:        INTEGER;
    boxRect:         Rect;
    globalPt:            Point;
    default:         INTEGER;
    saveTable:       MCTableHandle;
    ctlData:         CtlDataHdl;

BEGIN
    { lock control handle before dereferencing }
    HLock(Handle(theCntl));
    with theCntl^^ do begin
        { set hiliting to titlePart }
        ctrlHilite := titlePart;
        { invert the title rect }
        DrawTitle(theCntl);
        { get the pop-up box's rectangle }
        GetBoxRect(theCntl, boxRect);
        { get the topLeft point, and convert to global  }
        SetPt(globalPt, boxRect.left, boxRect.top);
        LocalToGlobal(globalPt);
        { get a handle to the pop-up menu }
        ctlData := CtlDataHdl(ctrlData);
        popMenu := ctlData^^.popMenu;
        { determine the default item }
        if (ctrlMax = popMenu^^.menuID) then
            default := ctrlMin
        else
            default := 1;
    { let the Menu Manager do the hard stuff }
        with globalPt do begin
            menuResult := PopUpMenuSelect(
                            popMenu,
                            v, h + 1, default);
        end;

        { what menu was the selection made from? }
        menuID := HiWord(menuResult);
        menuItem := LoWord(menuResult);
        { was a menu selection made? }
        if ((menuID <> 0) and ((menuID <> ctrlMax)
                or (menuItem<>ctrlMin))) then begin
            { check the current selection }
            if (BitAnd(vcLong, mCheck) =
                            mCheck) then begin
                { unmark previous selection }
                SetItemMark(
                    GetMHandle(ctrlMax),
                    ctrlMin,
                    CHR(noMark));
                { mark current selection }
                SetItemMark(
                    GetMHandle(menuID),
                    menuItem,
                    ctlData^^.markChar);
            end;  { if mCheck }
            { update the MenuSelect() results }
            ctrlMax := menuID;
            ctrlMin := menuItem;
            { redraw the pop-up box }
            DrawPopBox(theCntl, vcLong);
        end;  {if selection made }
    end;  { with }

    { unlock control handle before returning }
    HUnlock(Handle(theCntl));
END;  { doAutoTrack }

END.  { PopMenuCDEF.p }

/*******************************************************
Pop-up Menu Control "Rez" File Constants
'#include' this file in any Rez source file that uses
    pop-up menu controls.
*******************************************************/
/* standard pop-up menu control procID   */
#define  popMenuProc        48 /* #48 dec, $30 hex    */
/* CDEF's resource ID*/
#define pmCDEFResID        3
/* VARIATION CODE MODIFIERS: */
#define  mUnused            1   /* allow sub-menus    */
#define  mRes          2    /* add res names         */
#define  mCheck             4   /* check menu item    */
#define  mKey          8    /* reserved  */
/* parent of a sub-menu     */
#define  parent        "\$1B"
/* MARK CHARACTERS */
#define  cmdChar       "\$11"   /* command mark       */
#define  checkChar     "\$12"   /* check mark         */
#define  diamondChar   "\$13"   /* diamond mark       */
#define  appleChar     "\$14"   /* apple mark         */
/*******************************************************
END OF FILE:  PopMenusCDEF.r
*******************************************************/
```

Pascal Procedures
Fun With Regions, Part II

Stephen Dubin, V.M.D., Ph.D.,
Thomas W. Moore, PH.D.
Drexel University
Sheel Kishore, MS
MacTutor Vol. 4 No. 9

In our previous paper (Fun with Regions Part I: High Level Language Implementation), we showed how it was possible to estimate the area of an arbitrarily drawn region from high level languages such as Pascal and C using repeated application of the ROM subroutine, PtInRgn. Although this approach is simple and intuitive, execution time is excessive for large or complex regions. Whenever part of a high level language routine consumes an inconveniently long execution time, the possibility of using assembly language to achieve better efficiency should be considered. Two fundamental approaches may be applied. A study of the code generated by the compiler may reveal unnecessary looping, inefficient use of registers or other complexities which can be streamlined. If savings can be made within loops executed many times, the resulting speedup can be significant. The second approach applies when the task at hand is relatively simple and straightforward. In this case, determination of a specific efficient algorithm is the key, with translation to assembly code following directly. In dealing with the area computation part of our regions manipulation, we have explored both of these approaches and found the latter approach to be clearly superior.

In the first article, we provided the C Language code for the area calculation program at the end of the article with major subroutines interpolated into the text as implemented in Pascal. In this installment, the "tables are turned"; the interpolated routines are implemented for the Megamax C development system and the Consulair (MDS) assembly language system [This is included in the source code due to space considerations-ED]. At the end of the article, a program using the most useful of the assembly language optimizations is shown for Turbo Pascal with explanation of minor changes needed for the TML Pascal Development system. We recognize that our program may not be the most elegant or efficient approach to the problems; but even where an attempt at optimization yielded poor or marginal results, an interesting and - hopefully - useful technique is explored.

Several authors (Morton, M.: Reduce Your Time in the Traps! MacTutor October 1986 pp 21-24. and Knaster, S.: "How to Write Macintosh Software." Hayden, Hasbrouck Heights NJ, 1986, p 368) have advocated bypassing the trap dispatcher as a means of speeding routines in which ROM calls are made repeatedly. Certainly our CountPix routine, since it calls PtInRgn for every point within the region bounding box, is a candidate for this type of optimization. Mike Morton presents the underlying mechanism for this strategy, points out some cautions and pitfalls and shows how to do this task in Pascal using INLINE calls. Briefly, any call to the ROM must go through an intermediate step of finding the "true" address of the call in the particular version of the ROM in your machine before it can be invoked. When a ROM call is to be used many times, this address may be determined one time by means of the GetTrapAddress function early in the program; then you may employ some means of jumping to this address directly whenever the particular ROM call is to be used. In C language, the address might be acquired as follows:

```
trap = gettrapaddress(0xa8e8); /* trap is a global long integer */
                    /* A8E8 is the trap # for ptinrgn */
```

In order to see how the jump might be made, consider the following "glue" routine which is used for ptinrgn by the Megamax system:

```
boolean ptinrgn(pt, rgn)  /* as copied from qd13.c  */
point *pt;
rgnhandle rgn;
{
    asm {
  subq #2,A7          /* make room on the stack for the
  result */
  move.l pt(A6),A0    /*  address of point into A0 */
  move.l (A0),-(A7)   /*  dereference and put onto
  stack*/
  move.l rgn(A6),-(A7)/* region handle onto stack */
  dc.w 0xa8e8         /* call the ROM for ptinrgn */
  move.b (A7)+,D0 /* result into D0 where C expects to
  find the answer */
  ext.w D0            /* sign extend the result */
    }
}
```

Note that with Megamax inline assembly, the compiler takes care of setting up (and tearing down) the stack frame. Automatic (local) variables are accessed using the name of the variable as a displacement from A6. Global variables are treated similarly as offsets from A4. Thus if we have safely installed the true ROM address of ptinrgn in "trap", we can write a "new improved" version of the glue routine as follows:

```
boolean zptinrgn(pt, rgn)        /* same as ptinrgn except */
point *pt;                    /*  bypasses the trap dispatcher
  */
rgnhandle rgn;
{
    asm {
  subq #2,A7            /* make room on the stack for the
  result */
  move.l pt(A6),A0      /* address of point into A0*/
  move.l (A0),-(A7)     /* dereference and put onto stack */
  move.l rgn(A6),-(A7)  /* region handle onto stack */
  move.l trap(A4),A2        /* address of "true address" of
  ptinrgn into A2 */
  jsr (A2)          /* dereference once and jump there */
```


© The Definitive MacTutor, Vol. 4

```
    move.b (A7)+,D0        /* result into D0 where C expects to
    find the answer */
    ext.w D0               /* sign extend the result */
      }
}
```

When this version was used in place of ptinrgn, the time needed to estimate the area of a region was decreased by 15% for small simple regions and about 9% for larger and more complex ones. Although this would ordinarily be considered a significant improvement, it is little comfort to know that a five minute computation can now be completed in only four and a quarter minutes.

Upon examination of the disassembled code for counbtpix(), we noticed that the most often used variables were the points delimiting the region bounding box as well as the "exploring" point on which we called ptinrgn. Following classical optimization strategy, the next step was to set up these data structures on registers rather than to fetch them every time the coordinates of the exploring point were incremented. The code for this implementation of the countpix function is shown below:

```
bcountpix(theregion)/* sets up test point on registers */
rgnhandle   theregion;
{


   asm(

   move.l   trap(A4),A2       ; address of ptinregn
   move.w   #0xA8E8,D0        ; trap number for ptinrgn
   dc.w     0xA146            ; call the trap, address is in A0
   move.l   A0,A2             ; put it into A2

   move.l   theregion(A6),A3  ; regionhandle
   move.l   (A3),A1           ; dereference once
   move.l   2(A1),D4          ; topleft of rgnbox
   move.l   6(A1),D5          ; botright of rgnbox
   move.l   D4,D6             ; copy of TL as VH current point

hortest:
   cmp.w    D5,D6             ; compare horizontal
   blt.s    vertest               ; go on
   swap D4                 ; D4 is now HV
   addq.w   #1,D4             ; down 1 row
   swap D4                 ; now D4 is back to HV
   move.l   D4,D6             ; make this the current test point

vertest:
   swap D6                 ; now is HV
   swap D5                 ; now is RB
   cmp.w    D5,D6             ; compare vertical
   blt.s    pointest              ; go on
   bra.s    done                  ;

pointest:
   swap D5                 ; back to BR
   swap D6                 ; back to VH
   subq #2,A7                 ; make room for result
   move.l   D6,-(A7)          ; point onto stack
   move.l   theregion(A6),-(A7)   ; rgnhandle onto stack
   jsr (A2)                   ; go to ROM
   move.b   (A7)+,D0          ; result onto stack
   tst.b    D0                    ; was it true?
   beq skip                   ; not this time
   addq.l   #1,numpix(A4); yes, increment the counter
skip:
   addq.w   #1,D6             ; over 1 column
   bra.s    hortest               ; back for another point

done:
   }
}
```

As with the previous attempt at optimization, the speed increase with this approach was marginal at best. Our final attempt in this direction was to examine the code for ptinregn in the ROM in order to transpose (plagiarize?) it directly into the above routine. The result was surprising as well as disappointing. Although there was a measurable but tiny improvement for small simple regions, ones for which optimization was not needed anyway, the time needed to calculate the area of large, complex or disjoint regions increased significantly!! Our theory as to why this happens is based on the way in which the 68000 accesses ROM and RAM. Accesses to RAM (where the program resides) are shared with the video display, sound generator and disk speed controller. This leads to a RAM access rate of approximately six

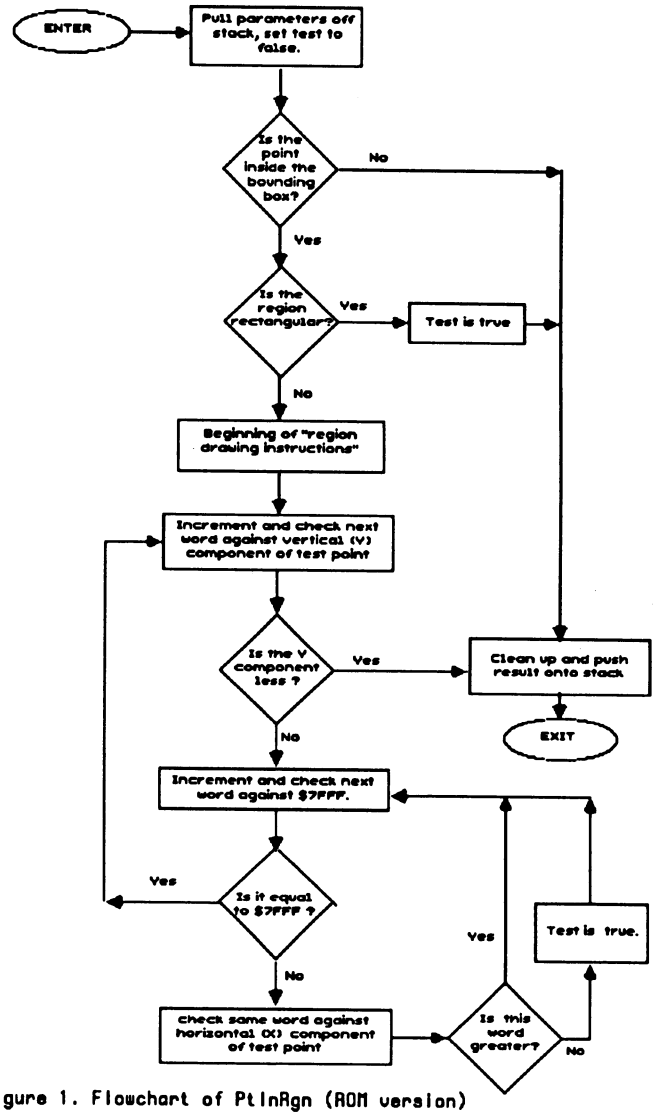
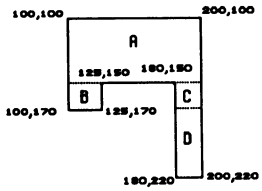


Figure 1. Flowchart of PtInRgn (ROM version)

megahertz. The ROM has a "direct line" and is accessed at 7.83 MHz (Inside Macintosh. III-18, Addison-Wesley, Reading MA, 1985).

All of this preoccupation with ptinregn led to an understanding of why the area computation takes so long for large or complex regions. A flow chart of how ptinregn works is shown in figure 1. Unless the region under examination is rectangular, it may be necessary to examine all of the region data, one word at a time, before deciding whether the point is indeed in the region. This is particularly true as the exploring point moves toward increasing values in the vertical (y) component. Clearly, our original countpix procedure, which calls ptinregn on every point in the bounding box, covers the same ground many times. Based on a conviction that the region information should be adequate to permit estimation of the area with one pass, we resolved to implement a specific algorithm "from the ground up."

As mentioned in the first installment of this article, region information is stored in memory in a way designed to require minimal space. A clear understanding of this method of encoding region boundaries is necessary in order to design our area calculating algorithm. To illustrate this process, consider the simple region plotted in Figure 2. The numbers on the plot are the coordinates of the "corners" of the outlined region. A memory dump of the data representing this region is shown below the graph. In order to design an algorithm for area calculation, we must understand the method of encoding the region in memory. As explained in our previous paper, the first five words of this data list are the data size in bytes (44) followed by the "upper left" and "lower right" coordinates of the rectangular boundary of the region - the regnbbox (100,100,220,200). Following these five words we find the information needed to compute area. Only horizontal boundary information is stored. The region being defined consists of the (rectangular) area under a given boundary line, extending down to the next horizontal boundary line encountered. Horizontal boundary lines are indicated by a y coordinate word, followed by start and stop x values (more than one pair if the line has multiple segments). The flag word, #7FFF, marks the end of the boundary segments at a particular y value. Therefore, the sixth word of data (100) is the y coordinate of the top boundary line of the region. It is followed by x values 100 (start of line) and 200 (end of line). Then we get a flag indicating that no more boundary segments exist at this level. The next horizontal boundary is the line from (125,150) to (180,150).



Fun with Regions II

Here are the first 400 words of the region data. (FLAG = 32767)

```
 44   100   100   220   200   100   100   200 FLAG   150   125   180 FLAG   170   100   125
FLAG   220   180   200 FLAG FLAG
```

Figure 2. "Example" region along with a screendump of the region data.

Figure 2

Therefore we find 150 (y value) and 125,180 (x start and stop values) to be the next three words of the data. In this manner, the remaining data can be seen to define the region of figure 2. The double flag indicates the end of the data table.

The question now is how to use this table to calculate the area of the region thus defined. The arrangement of the data suggests dividing the area into rectangular pieces and adding their areas. We might start by subtracting the first x value (100) from the second (200) to get the width of the top of the first rectangle (marked "A"). The y value of this line could be put aside to be subtracted from the next y value (170) yielding the height of the rectangle. The product of these dimensions is one component of the final area

The x values following the 150 are endpoints of a new horizontal boundary line. Since this line falls under the previous boundary, it represents the bottom of a rectangular piece rather than the top of a new one. From here (y = 150) our region will now grow downward in two rectangular pieces, B and C. To calculate the areas of these pieces, we must make use of the two new x values found in the data table (125,180). If we arrange all x values found so far in order of magnitude (100, 125, 180, 200), the appropriate widths can be found by pairing the values and subtracting the first from the second in each pair. This is a rule we can use in our algorithm: maintain an ordered list of all encountered x values, pair them and subtract the first of each pair from the second. The sum of these differences will be the total width at the top of each of the rectangular regions. The y coordinate at the top of B and C is subtracted from the next y coordinate found (170-150) to determine the height of these rectangles. Height times width is then added to the accumulating total area.

One problem remains: how to end the process? Following y value 170, we find x values of 100 and 125 in the data. One leg of our descending area ends here so we would like our x table to list just 180, 200 (the top of D) from here on. Therefore, the final rule we need for our area algorithm is to remove entries from the x value ordered list whenever they are matched by a newly encountered x value. At the final y value (220), we subtract the remaining x pair (200-180), and multiply by the last y difference (220-170), giving the area of the last piece, D.

A flowchart of this process is given in figure 3. The routine as implemented to provide a linkable object file using the MDS (now Consulair) 68000 assembly system is shown below. Using the TTAA (Tom Terrific Area Algorithm), even the largest and most complex region that could be drawn on the Macintosh screen could have its area estimated in less than 20 seconds. Such regions take as long as ten minutes using the old CountPix.

This is the code for use with the MDS assembler to produce the file ACountPix.REL. This can then be linked to a Pascal or other "main" program.

```
;
; ACountPix.asm
; Pascal Usage: Function ACountPix( theRegion:RgnHandle) :
LongInt;
; This function emulates CountPix
; Written by Thomas W. Moore, Ph.D. and Stephen Dubin, V.M.D.,
Ph.D.
```

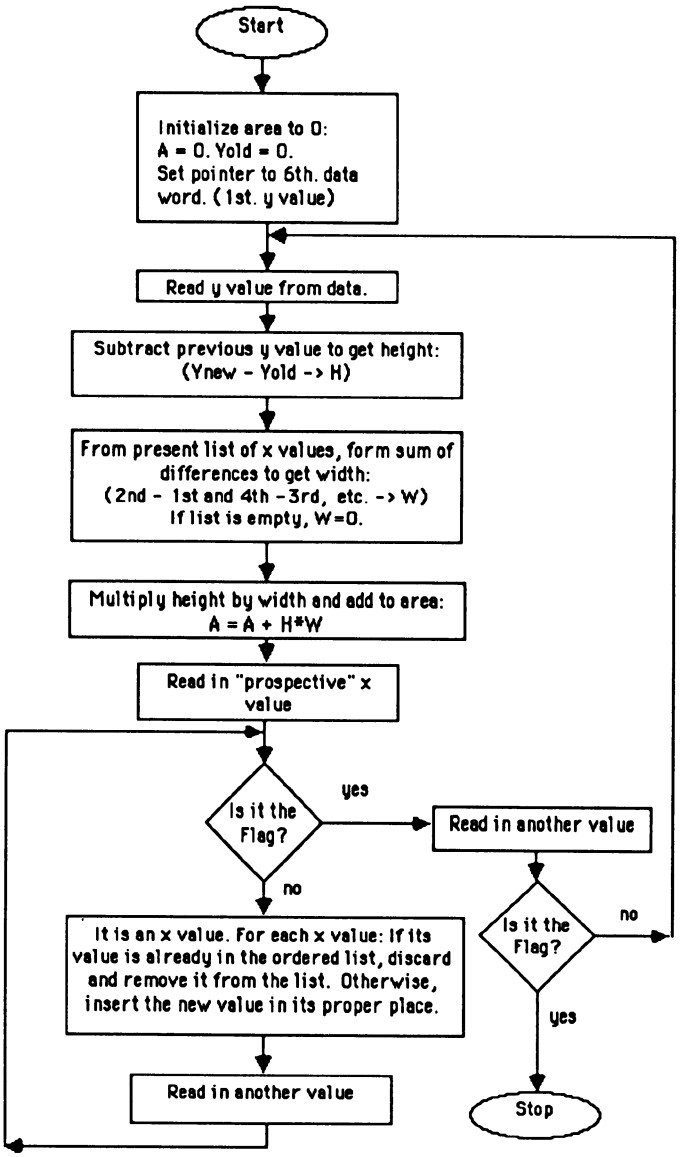


Figure 3. Flowchart of "Tom Teriffic" Area Algorithm

```
; Copyright © 1987

            XDEF  ACountPix
            XREF  myBUF
; The buffer is allocated in the calling program even though it might be
; more elegant to allocate it here with a DS statement; however Turbo
; Pascal V1.0 seems intolerant of this. ( p 335 of the Turbo Pascal manual)
;————————————— INCLUDES —————————————

Include     Traps.D          ; Use System and ToolBox traps
Include     ToolEqu.D        ; Use ToolBox equates



ACountPix:
  link      A6,#0            ; set up frame pointer
  movem.l   A0-A3/D0-D7,-(A7)  ; save the world
  clr.l     -(A7)            ; make room on stack for result
  movea.l   8(A6),A0         ; region handle into A0
  movea.l   (A0),A0          ; dereference => pointer in A0
  clr.l     D7               ; set area to zero
  lea       myBUF(A5),A1     ; lowest address of x list

rectcheck:                   ; see whether it is a rect and if so - do  the job here
  cmpi.w    #10,(A0)         ; is this a single rectangle
  bne.s     morework         ; if not do the big job
  move.w    4(A0),D1         ; left
  move.w    8(A0),D2         ; right
  move.w    2(A0),D3         ; top
  move.w    6(A0),D4         ; bottom
  sub.w     D1,D2            ; width
  sub.w     D3,D4            ; height
  mulu.w    D2,D4            ; area in D4
  move.w    D4,D7            ; lower word into D7
  bra       done

morework:                    ; get ready for some serious work
  lea       10(A0),A0        ; beginning of region info
  clr.l     D4               ;
  clr.l     D2               ;
  move.l    #512,D3          ; size of buffer to hold ordered list of x values
  adda.l    D3,A1            ; highest address in buffer
  movea.l   A1,A2            ; copy in A2
  movea.l   A1,A3            ; another in A3
  move.w    #-1,(A1)         ; -1 in highest x address so that 1st x entry will be greater

gety:                        ; read in y coordinate of next horizontalboundary
  move.w    (A0)+, D3        ; latest y value
  jsr       calc

getx:
  move.w    (A0)+,D1         ; new x value
  cmpi.w    #$7fff,D1        ; flag indicates no more x values at this y
  bne       storex           ; if no flag, it is a new x
  move.w    (A0),D1          ; next word of region info
  cmpi.w    #$7fff,D1        ; all done?
  beq       done             ; yes go home
  bra       gety             ; no, get next y

storex:                      ; place new x value in proper place in ordered list
  movea.l   A3,A1            ; A3 points to highest x value in ordered list
  cmp.w     (A1),D1          ; compare new x value to largest entry
  bne       s1               ; if not equal, it must be added to list
  addq      #2,A3            ; if match, remove from list
  bra       getx             ; next x

s1:
  lea       -2(A3),A3        ; add a space at high end of list for new x
  bgt.s     insert           ; if new x value is greatest, put it on top

mkroom:                      ; new x is not greatest so we must move list values up to make room
  move.w    (A1)+,-4(A1)     ; move data up (1 word net distance)
  cmp.w     (A1),D1          ; compare next list entry
  beq.s     remove           ; if it matches, remove it
  bcc.s     insert           ; it is greater, so put it above
  cmpa.l    A1,A2            ; are we at bottom?
  bne       mkroom           ; no, move another one up

insert:                      ; insert new x value in ordered place in list
  move.w    D1,-(A1)         ; insert above present location
```

```
  bra       getx            ;

remove:                     ; erases an entry from the list
  subq      #2,A1           ; point to next higher

r1:
  cmpa.l    A1,A3           ; is it the top?
  beq       shrink          ; yes so exit
  move.w    -(A1),4(A1)     ; move greater x values down to
            replace
  bra       r1              ; value removed

shrink:
  addq      #4,A3           ; if a match occurred, list shrinks
            by 2 words
  bra       getx            ; one that we didn't insert and one
            that we erased

calc:                       ; determine new Height
  sub.w      D3,D4          ; Y old - Y new
  neg.w     D4              ; Height of the rectangle(s)

newW:                       ; prepare for Width calculation
  clr.l     D2              ; Will receive width
  clr.l     D1              ; work reg
  movea.l   A2,A1           ; reset A1 to point to least x value
            in list

dx:                         ; check to see if all x pairs have
            been used.
                            ; multiply H x W and add to area
  cmpa.l    A1,A3           ; A3 points to greatest x value in
            list
  bne       morex           ; if not equal, not all x's have been
            used
  mulu        D4,D2         ; H x W
  add.l     D2,D7           ; add to accumulating area
  move.w    D3,D4           ; for next time
  rts

morex:                      ; subtracts x values in pairs adding
            differences to accumulating W
  move.w    -(A1),D1        ; Xi (lower x value of a pair)
  sub.w     -(A1),D1        ; Xi - Xi+1 (length of a horizontal
            boundary segment)
  neg.w     D1              ; Xi+1 - Xi (correct sign)
  add.w     D1,D2           ; W (add to accumulating width)
  bra       dx              ;

done:
  move.l    D7,12(A6)       ; store result "under" the last
            parameter
  movem.l   (A7)+,A0-A3/D0-D7  ; restore registers
  unlk      A6              ; restore original stack
  move.l    (A7)+,A0        ; get return address
  addq.l    #4,A7           ; remove parameters
  jmp       (A0)            ; return this way

  end
```

The same algorithm can be used with the very convenient inline assembly facility of the Megamax and other C development systems. Because these compilers take care of "tending the stack" for you, the entry and exit procedures are significantly simplified. For the Megamax systems, they are as follows:

```
acountpix(theregion)
rgnhandle       theregion;
{
int     buf[1000];
  asm(
  move.l theregion(A6),A0  ; regionhandle  note: local
variables are                    referred off A6
  move.l (A0),A0        ; dereference once => region pointer
  clr.l  D7             ; set area to zero
  lea    buf(A6),A1     ; lowest address of x list

 rectcheck:

/*  Everything in between is the same as in ACountPix.Asm
above */
done:
  move.l D7,numpix(A4)   ;report the answer note: global
variables                        are referred offA4

  }
}
```

This is the code for our main calling program as implemented in Turbo Pascal:

```
{PasArea.Pas }
{Copyright 1987 by Stephen Dubin, V.M.D.and Thomas W. Moore,Ph.D.
}
{Prepared with Turbo Pascal V1.0                                }
{ Users of other Pascal systems should particularly check the
"preamble"}
{ portion of their program (Linking directives, "uses", "in-
cludes", etc.)
{ also check usage of type "point" - TML doesn't like use of pt.h
and}
{ pt.v as control elements in a for statement.                  }

program PasArea;

{$R-}               { Turn off range checking                   }
{$I-}               { Turn off I/O error checking               }
{$R PasArea.rsrc}   { Identify resource file                    }
{$U-}               { Turn off auto link to runtime units       }
{$L ACountPix.Rel } { Link in Assembly Language Segment}
{$D+}               { Embed Procedure Labels                    }

uses  Memtypes,QuickDraw,OSIntf,ToolIntf,PackIntf;

const
  FileMenuID = 1;          { the File menu}
  OptionMenuID = 2;                { the option menu}
  WindResID = 1;           { the resource id of my window}

type
  BUF   = array[1..512] of Integer; { Make it bigger if you are
really paranoid}

var
  myMenus : Array[FileMenuId..OptionMenuID] of MenuHandle;
  Done : Boolean;
  MyWindow : WindowPtr;
  TotalRegion   :   RgnHandle;
  Numpix        :   Longint;
  myBUF         : BUF;

function ACountPix( theRegion:RgnHandle) : LongInt; external;

function CountPix(theRegion : RgnHandle): LongInt;

var
 pt : Point;
 rgn    :   Region;
 temp   :   LongInt;
 x      :   Integer;
 y      :   Integer;

begin
   temp   :=  0;
```

```
   rgn  :=  theRegion^^;
   for  x  := rgn.rgnBBox.left  to  rgn.rgnBBox.right do
        begin
             pt.h := x;
            for y := rgn.rgnBBox.top to rgn.rgnBBox.bottom do
                  begin
                  pt.v := y;
                if  PtInRgn( pt, TheRegion) then  temp := temp +
1;
                  end;
        end;
        CountPix := temp;
end;
{ Turbo seems to accept pt.h and pt.v as control elements but TML
does}
{ not. Some format checkers agree with TML}

procedure Wipe;
var
    r   :   Rect;

begin
    SetRect(r,0,0,504,300);
    EraseRect(r);

end;

procedure Data;
var
    rgn         :   Region;
    rgnpntr     :   Ptr;
    size        :   Integer;
    thebuf      :   BUF;
    bfpntr      :   Ptr;
    myString    :   Str255;
    i           :   Integer;
    x           :   Integer;
    y           :   Integer;

 begin
    Wipe;
    TextSize(9);
    TextFont(Monaco);
    rgn  :=  totalRegion^^;
    rgnpntr := ptr(totalRegion^);
    size := rgn.rgnSize;
    if size > 800 then size:= 800;
    bfpntr := ptr(@thebuf);
    BlockMove(rgnpntr,bfpntr,size);
    MoveTo(10,10);
   DrawString('Here are the first 400 words of the region data.
(FLAG = 32767)');
    x := 10;
    y := 20;
    for i  := 1  to  (size div 2) do
        begin
        MoveTo(x,y);
        NumToString(theBuf[i],myString);
        if theBuf[i] < 32766 then
            begin
                if theBuf[i] <10  then DrawString(' ');
                if theBuf[i] <100 then DrawString(' ');
                if theBuf[i] < 1000 then DrawString(' ');
                if theBuf[i] < 10000 then DrawString(' ');
                DrawString(MyString);
            end;
        if theBuf[i] > 32766 then DrawString(' FLAG');
        x := x + 30;
        if (i mod 16) = 0 then
            begin
            x := 10;
            y := y+10;
            end;
        end;
end;

procedure OvalRegion;
var
    RectA : Rect;

begin
   Wipe;
   TotalRegion := NewRgn;
   SetRect(RectA, 170,175,195,200);
   OpenRgn;
   ShowPen;
   FrameOval(RectA);
   HidePen;
   CloseRgn(TotalRegion);
end;

procedure Contour;
var
    p1  :   Point;
    p2  :   Point;
    OldTick :  Longint;

begin
  Wipe;
  TotalRegion := NewRgn;
  OldTick := TickCount;
  Repeat
    GetMouse(p1);
    MoveTo(p1.h,p1.v);
    p2 := p1;
  Until Button = True;
  OpenRgn;
  ShowPen;
  PenMode(patXor);
  Repeat
    GetMouse(p2);
    Repeat Until (OldTick <> TickCount);
    LineTo(p2.h,p2.v);
  Until Button <> True;
  Repeat Until (OldTick <> TickCount);
  LineTo(p1.h,p1.v);
  PenNormal;
  HidePen;
  CloseRgn(TotalRegion);
  InvertRgn(TotalRegion);
end;

procedure Example;

begin
    Wipe;
    OpenRgn;
    TotalRegion := NewRgn;
    ShowPen;
    MoveTo(100,100);
    LineTo(200,100);
    LineTo(200,220);
    LineTo(180,220);
    LineTo(180,150);
    LineTo(125,150);
    LineTo(125,170);
    LineTo(125,170);
    LineTo(100,170);
    LineTo(100,100);
    HidePen;
    CloseRgn(TotalRegion);
end;

procedure FreeBox;
var
    p1  :   Point;
    p2  :   Point;
    p3  :   Point;
```

```
    OldTick :  Longint;
    MyRect  :  Rect;

begin
    Wipe;
    TotalRegion := NewRgn;
    OldTick := TickCount;
    PenPat(gray);
    PenMode(patXor);
    Repeat
    GetMouse(p1);
    p2 := p1;
    Until Button = True;
    OpenRgn;
    ShowPen;
    PenMode(patXor);
    Repeat
    Pt2Rect(p1,p2,MyRect);
    Repeat Until (OldTick <> TickCount);
    FrameRect(MyRect);
        Repeat
            GetMouse(p3);
        Until  EqualPt(p2,p3) <> True;
   Repeat Until (OldTick <> TickCount);
   FrameRect(MyRect);
   p2 := p3;
   Until Button <> True;
   Pennormal;
   HidePen;
   PenPat(black);
   FrameRect(MyRect);
   CloseRgn(TotalRegion);
   InvertRgn(TotalRegion);
end;

procedure Area;
var
    NumTix  :   LongInt;
    MoreTix :   LongInt;
    TicString   :   Str255;
    PixString   :   Str255;

begin
   TextFont(Monaco);
   TextSize(9);
   TextMode(0);
   MoveTo(10,20); DrawString(' Using Pascal ');
   NumTix := TickCount;
   NumPix :=  CountPix( TotalRegion );
   MoreTix := TickCount - NumTix;
   NumToString(MoreTix,TicString);
   NumToString(NumPix,PixString);
   MoveTo(10,30); DrawString(' Tickcount = ');
   MoveTo(120,30); DrawString(TicString);
   MoveTo(10,40); DrawString(' Pixel Number = ');
   MoveTo(120,40); DrawString(PixString);
   MoveTo(10,50); DrawString(' Using Tom Terrific ');
   NumTix := TickCount;
   NumPix :=  ACountPix( TotalRegion );
   MoreTix := TickCount - NumTix;
   NumToString(MoreTix,TicString);
   NumToString(NumPix,PixString);
   MoveTo(10,60); DrawString(' Tickcount = ');
   MoveTo(120,60); DrawString(TicString);
   MoveTo(10,70); DrawString(' Pixel Number = ');
   MoveTo(120,70); DrawString(PixString);
end;

procedure ProcessMenu(codeWord : Longint);
var
  menuNum : Integer;
  itemNum : Integer;

begin
  if codeWord <> 0 then
    begin
      menuNum := HiWord(codeWord);
      itemNum := LoWord(codeWord);
      case menuNum of
                FileMenuID :Done := true;
        OptionMenuID :
                    begin
                case ItemNum of
                    1:Contour;      {Contour}
                    2:FreeBox;      {Freebox}
                    3:OvalRegion;   {Oval}
                    4:Example;      {Example}
                    5: Area;        {Area}
                    6:Data;         {Region Data}
                        end; { of ItemNum case}
       end;{ of MenuNum case}
    end;
  HiliteMenu(0);
 end;
end;

procedure DealWithMouseDowns(theEvent: EventRecord);
var
  location : Integer;
  windowPointedTo : WindowPtr;
  mouseLoc : point;
  windowLoc : integer;
  VandH : Longint;
  Height : Integer;
  Width : Integer;

 begin
  mouseLoc := theEvent.where;
  windowLoc := FindWindow(mouseLoc,windowPointedTo);
  case windowLoc of
    inMenuBar :
      begin
        ProcessMenu(MenuSelect(mouseLoc));
      end;

  end;
end;

procedure MainEventLoop;
var
  Event : EventRecord;
  theItem : integer;

begin
  repeat
    SystemTask;
    if GetNextEvent(everyEvent, Event) then
     begin
              case Event.what of
               mouseDown : DealWithMouseDowns(Event);
              end;
     end;
  until Done;
end;

procedure MakeMenus;
var
  index : Integer;
begin
  for index := FileMenuId to OptionMenuID do
    begin
      myMenus[index] := GetMenu(index);
      InsertMenu(myMenus[index],0);
    end;
  DrawMenuBar;
end;

{       Main Program       }
```


© The Definitive MacTutor, Vol. 4

```
begin
  Done := false;    FlushEvents(everyEvent,0);
  InitGraf(@thePort);        InitFonts;        InitWindows;
  InitMenus;        InitDialogs(nil);
  InitCursor;
  MoreMasters;
  MoreMasters;
  MakeMenus;
  MyWindow := GetNewWindow(WindResID,nil,Pointer(-1));
  SetPort(MyWindow);
  TotalRegion := NewRgn;   {Lazy way to avoid bomb if your select
"Area" first}
  MainEventLoop;
end.
```

Here is the resource file for use with the above program (Turbo Pascal):

```
*
*   Resource listing from file: "PasArea.R".
*

PasArea.rsrc

Type AREA = STR
,0
PasArea, by Stephen Dubin and Thomas W. Moore Copyright © 1987

Type WIND
,1
Fun with Regions II
40 5 330 505
Visible NoGoAway
0
0

Type MENU
,1
File
Quit

,2
Option
Contour
Freebox
Oval
Example
Compute Area
Region Data
```

In order to compile the same program with TML Pascal V2.0, a few minor adjustments were needed. The preamble was changed to:

```
program TMPasArea;

{$T APPL AREA}    { set the type and creator}
{$B+}    { set the bundle bit}
{$L TMPasAreaRes} { link the resource file too...}

uses MacIntf;

{ Constant, Type and Variable declarations as above are the same
as in PasArea.Pas above}

{       Declare the Assembly Language routine as external }
function ACountPix( theRegion:RgnHandle) : LongInt; external;
{$U ACountPix     }
{ This directive will not appear in the .link file unless it follows
the declaration of the }
{ relocatable object file as external}
```

The only change needed in the body of the program was in the high level CountPix function. A form that compiled with TML is:

```
function CountPix(theRegion : RgnHandle): LongInt;

var
 pt : Point;
 rgn    :   Region;
 temp   :   LongInt;
 x      :    Integer;
 y      :    Integer;

begin
   temp   :=  0;
   rgn  :=  theRegion^^;
   for  x  := rgn.rgnBBox.left  to  rgn.rgnBBox.right do
        begin
            pt.h := x;
            for y := rgn.rgnBBox.top to rgn.rgnBBox.bottom do
                begin
                   pt.v := y;
                  if  PtInRgn( pt, TheRegion) then  temp := temp
+ 1;
                end;
        end;
        CountPix := temp;
end;
```

TML does not seem to like having pt.h and pt.v as control elements. PasMat, a Pascal formatting and syntax checking program, agrees with TML on this point. In keeping with our local traditions, the first non-comment line of our TML resource file was "TMPasAreaRes". Although it probably is of little interest in these days of monstrous memories, the TML version of the program requires 3,305 bytes of memory; whereas the Turbo program weighs in at a hefty 10,855 bytes.

Some final zingers for the reader - Although it was certainly necessary for us to use assembler to plumb the depths of the ROM and to work out the algorithm for making our area measurement lightning fast; one might consider whether the same algorithm might now be implemented entirely from C, Pascal or possibly Basic. Would the speed be degraded to any appreciable extent? Will a new call AreaRgn be found in the 512K Roms on the Mack III's?

Programmer's Workshop

Menus In Windows

James Matthews
Hanover, NH
MacTutor Vol. 4 No. 11

[Jim Matthews is a software developer at Dartmouth, working on network applications. He has done maintenance work on DarTerminal, an AppleTalk terminal emulator, and has worked on developing a Macintosh mail system. He started programming in high school on an IBM 360 with core memory but slowly moved to smaller, newer machines. In college, he helped develop MacFunction, a three-dimensional graphing program that is currently marketed by True Basic, Inc.]

Menus in Windows
by Jim Matthews, Dartmouth College

One of the critical elements of the Macintosh user interface is the menu bar. It relieves the user of the need for memorizing command names and gives easy access to a large number of operations in a small amount of space. Nonetheless, the limitations of the standard Macintosh menu bar have become increasingly evident. Large screens make the fixed position of the menu bar less convenient, and large programs present users with an overwhelming number of available commands. Desk accessories have never been able to use the menu bar fully, limiting their potential. Both Apple and third parties have developed workarounds to the limitations of the menu bar. Hierarchical and pop-up menus, implemented in System 4.1, provide alternate ways of structuring menu commands. The tear-off palettes used in HyperCard and MacPaint 2.0 add a new dimension to menus, as do the tear-off menus provided with Radius displays.

Still, I recently felt the need to extend the menu bar concept by another step. While developing a program with a large number of commands, I decided that what I wanted was a different menu bar for each window. It would have been possible to change the menu bar depending on which window was in front, but the commands needed by each window were so different that the user would never know what to expect when he pulled down a menu. Furthermore, I wanted to leave open the possibility of turning the program into a desk accessory, and I could not fit all my commands into one menu. So I re-implemented part of the Menu Manager to provide for window-specific menu bars. The code itself is not very complicated — I was fortunate to be preceded by Mike Schuster, whose December, 1985 article on pop-up menus provided a wealth of useful information. The routines I produced met my needs admirably, but like any extension to the standard interface they also raised some tricky issues.

The wMenu Manager Routines

Because I wanted to produce menu bars that function exactly like the one at the top of the screen, I implemented the window menus by imitating eight standard Menu Manager routines. They all operate on a wMenuBar data structure which is roughly comparable to a MenuList.

```
type
wMenuRec = record
  mh : MenuHandle;
  titleRect : Rect;
  end;
wMenuBar = record
  numMenus : integer;
  hilited : integer;
  gp : GrafPtr;
  wMenus : array [0..0] of wMenuRec;
  end;
```

The wMenuBar stores the number of menus in a menu bar, which one is hilited (if any), and the GrafPort in which the menu bar is drawn. In addition, it stores a menu handle and a rectangle for each menu that has been inserted. The rectangle specifies the coordinates of the menu title; i.e. the area that is inverted when a menu is selected. This is a bit wasteful, since two of the rectangle's coordinates are always the same, but it makes the code simpler. The wMenuBar record is created by a call to wInitMenus, which allocates the storage and returns a wMenuBarHandle. The rest of the routines accept the same arguments as their Menu Manager equivalents, with the addition of a wMenuBarHandle to specify the menu bar being altered. The routines are wInsertMenu and wDeleteMenu, to add and remove menus from the menu bar; wClearMenuBar to delete all the menus; wDrawMenuBar to redraw the menu bar; wMenuSelect and wMenuKey to respond to mouse and key events, respectively; and finally, wHiliteMenu to highlight a menu's title.

Implementation Issues

There are a few subtle points in the implementation of the

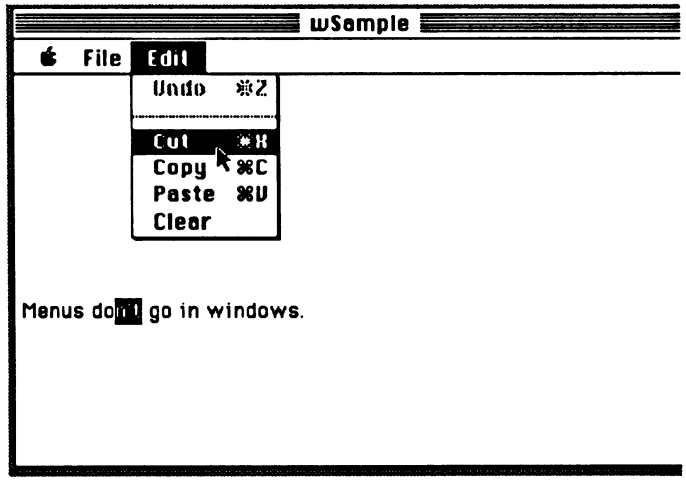


Figure 1. Edit Menu

© The Definitive MacTutor, Vol. 4

wMenu Manager. The System 4.2 menu definition procedure has a bug that is fixed by initializing the low memory global TopMenuItem in wInitMenus. Calls to the menu definition procedure are implemented using inline machine code, since Pascal doesn't provide a standard way to call a routine based on its address. WMenuKey walks down a menu's data looking for a certain command key equivalent, and the dynamic nature of the data structure requires some ugly code. GetNextEvent and SystemTask are called in the inner loop of the menu selection code: this means that keyDown events are swallowed while the user is holding down a menu, but it provides the ability to produce screen dumps and keeps desk accessories updated. With the standard menu bar it is impossible for the user to drag the mouse above the top of the menus, but with window menus it was necessary to deal with this case. I decided to have the displayed menu disappear when the user moved the mouse above a window's content region, but that could easily be changed.

The wMenu routines can be substituted for Menu Manager ones with a few exceptions. Unlike InitMenus, wInitMenus must not be called until there is a window to put the menu bar in. wDrawMenuBar should be called in response to update events, since the Window Manager considers the menu bar part of a window's content region. This also means that wMenuSelect should be called in response to mouseDown events in a window's content region. A program should continue call MenuSelect and MenuKey at appropriate times to give the user access to desk accessory menus.

The wMenu routines make it fairly easy to implement a number of different menu bars in one program, but that can lead to an explosion of possible commands. The nested case statements that typically handle menu commands can become unwieldy when the number of menu bars exceeds two or three.

Interface Issues

Menus in windows have the significant disadvantage of being a departure from the standard way of doing things. Users do not expect to find a menu bar inside a window, and can be confused by one. If there is more than one menu bar visible the user may wonder which to use, and if the user types a command

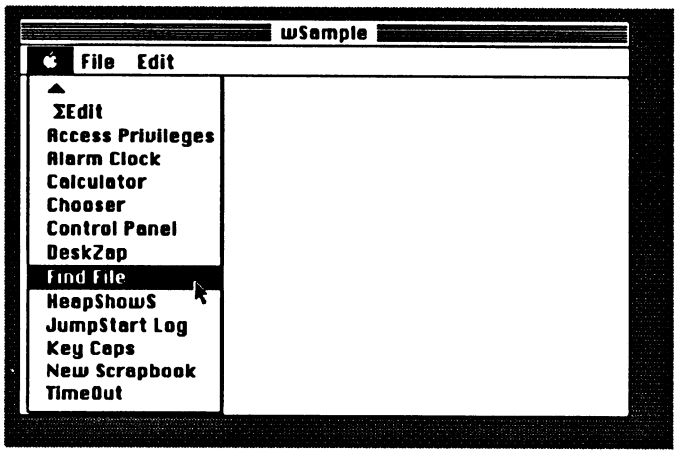


Figure 2. Scrolling Menu

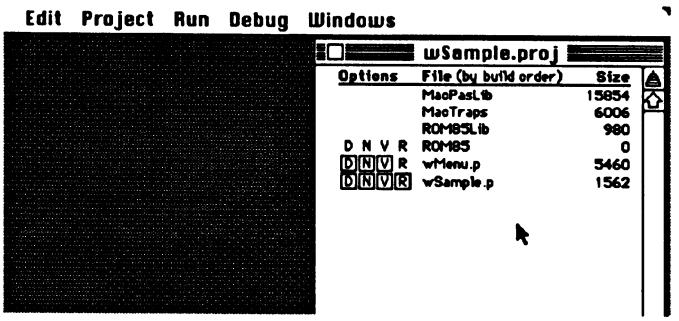


Figure 3. Project Window

key equivalent it may not be clear which menu bar is handling it. This problem becomes especially difficult when some commands are available from the regular menu bar and others only from window menus. I have tried to use window menus for commands that only affect that particular window, and the regular menu bar for program-wide operations. This introduces a degree of modality into a program, but not much more than is present in any multiple-window application.

Compatibility Issues

The wMenu routines were written to be compatible with all post Mac 512 models. The 128k ROM calls HSetState and HGetState are used in a few places; but if they were changed, I imagine the code would work on older Macs also. Given Apple's emphasis on the sanctity of the Window Manager's GrafPort, care was taken to make sure that menus are only drawn inside an application window's content region if they overlap they either scroll or are cropped. There are a couple of things that could cause future compatibility problems, though. First, the program modifies TopMenuItem, and although the fix was made on the advice of Mac DTS, it could break on future systems. Secondly, Apple is now asking programmers not to rely on the internal structure of menu records. The wMenuKey routine could not be written without doing this, so it is vulnerable to future changes. The code emulates the pre-Mac SE Menu Manager in that color and hierarchical menus are not supported. The code has been tested on machines from the 512KE to the Mac II without mishap.

wSample

I have included a wMenu version of the sample program found in Inside Macintosh Vol. I. This example illustrates how the wMenu routines are called and demonstrates how the menus appear to the user. It is not, however, a case where window menus add much to the program. Window menus are much more appropriate when there are multiple windows, each with significantly different functionality. I originally implemented them as part a mail program that had separate windows for composing letters, reading mail, and organizing a mailbox. In that case, window menus let me compartmentalize the program's interface and use a large number of commands without overwhelming the standard menu bar.

```
{ wMenu Manager }
{ by Jim Matthews }
```

```
UNIT wMenu;

INTERFACE

  USES
    ROM85;   { uses HGetState and HSetState }

  CONST
    mBarHeight = 20;
    betweenTitles = 15;
{ # of pixels between adjacent menu titles }
    invertOverlap = 10;
{ # of pixels to invert on each side of a menu title }
    noneHilited = -1;
{ value to store in wMenuBar.hilited if nothing hilited }
    menuTitleBit = 31;
{ mac-style bit offset for menu title bit in enableFlags }

  TYPE
    wMenuRec = RECORD
        mh : MenuHandle;
        titleRect : Rect;
      END;
    wMenuBar = RECORD
        numMenus : integer;
        hilited : integer;
        gp : GrafPtr;
        wMenus : ARRAY[0..0] OF wMenuRec;
      END;
    wMenuBarPtr = ^wMenuBar;
    wMenuBarHandle = ^wMenuBarPtr;

  FUNCTION wInitMenus (gp : GrafPtr) : wMenuBarHandle;
  PROCEDURE wInsertMenu (theMenuBar : wMenuBarHandle;
                    theMenu : MenuHandle;
                    beforeID : integer);
  PROCEDURE wDrawMenuBar (theMenuBar : wMenuBarHandle);
  PROCEDURE wDeleteMenu (theMenuBar : wMenuBarHandle;
                    menuID : integer);
  PROCEDURE wClearMenuBar (theMenuBar : wMenuBarHandle);

  FUNCTION wMenuSelect (theMenuBar : wMenuBarHandle;
                    startPt : Point) : longint;
  FUNCTION wMenuKey (theMenuBar : wMenuBarHandle;
                    ch : char) : longint;
  PROCEDURE wHiliteMenu (theMenuBar : wMenuBarHandle;
                    menuID : integer);

IMPLEMENTATION

{wInitMenus - create a wMenuBar and associate it with a
grafport}
  FUNCTION wInitMenus;  { (gp : GrafPtr) : wMenuBarHandle; }
    TYPE
      intptr = ^Integer;
    VAR
      mbar : wMenuBarHandle;
      TopMenuItemP : intptr;
  BEGIN
    { Set low-mem global to fix menu display bug }
    TopMenuItemP := intptr($A0A);
    TopMenuItemP^ := mBarHeight;

    mbar := wMenuBarHandle(NewHandle(sizeof(wMenuBar)));
    mbar^^.numMenus := 0;
    mbar^^.hilited := noneHilited;
    mbar^^.gp := gp;
    wInitMenus := mbar;
  END; { wInitMenus }

{wInsertMenu - insert a menu into a defined wMenuBar,}
{ analogous to InsertMenu }
  PROCEDURE wInsertMenu; { (theMenuBar : wMenuBarHandle; }
                    { theMenu : MenuHandle;
                    { beforeID : integer); }
    VAR
      newSize : Size;
      r : Rect;
      i, j : integer;
      titleWidth, oldSize, oldFont : integer;
  BEGIN
    newSize := sizeof(wMenuBar) + theMenuBar^^.numMenus *
sizeof(wMenuRec);
    IF newSize > GetHandleSize(handle(theMenuBar)) THEN
      SetHandleSize(handle(theMenuBar), newSize);
    IF MemError = noErr THEN
      BEGIN
        oldSize := thePort^.txSize;
        oldFont := thePort^.txFont;
        TextSize(12);
        TextFont(systemFont);
        titleWidth := StringWidth(theMenu^^.menuData);
        TextSize(oldSize);
        TextFont(oldFont);
        i := 0;
        IF beforeID > 0 THEN
        { Insert the menu before a particular one? }
          BEGIN
            i := 0;
            WHILE (theMenuBar^^.wMenus[i].mh^^.menuID <>
beforeID) AND (i < theMenuBar^^.numMenus) DO
              i := i + 1;
            IF i <> theMenuBar^^.numMenus THEN
              BEGIN
                FOR j := theMenuBar^^.numMenus DOWNTO i + 1
DO
                  BEGIN
                    theMenuBar^^.wMenus[j].mh :=
theMenuBar^^.wMenus[j - 1].mh;
                    theMenuBar^^.wMenus[j].titleRect :=
theMenuBar^^.wMenus[j - 1].titleRect;

OffsetRect(theMenuBar^^.wMenus[j].titleRect, titleWidth +
betweenTitles, 0);
                  END; { for loop - copying menus back }
              END; { if there's a menu id = beforeID }
          END { if beforeID <> 0 }
        ELSE     { if beforeID <= 0, put it after the rest }
          i := theMenuBar^^.numMenus;
        WITH theMenuBar^^.wMenus[i] DO
          BEGIN
            titleRect.top := 1;
            titleRect.bottom := mBarHeight - 1;
            IF i = 0 THEN
              titleRect.left := betweenTitles - invertOverlap
            ELSE
              titleRect.left := theMenuBar^^.wMenus[i -
1].titleRect.right + betweenTitles - 2 * invertOverlap;
            titleRect.right := titleRect.left + titleWidth +
2 * invertOverlap;
            mh := theMenu;
          END; { with theMenuBar^^.wMenus[i] }
        theMenuBar^^.numMenus := theMenuBar^^.numMenus + 1;
      END; { no MemError }
  END; { wInsertMenu }

{ wDrawMenuBar - draw the wMenuBar, with appropriate}
{ highlighting }
  PROCEDURE wDrawMenuBar; { (theMenuBar : wMenuBarHandle); }
    VAR
      i : integer;
      r : Rect;
      oldPort : GrafPtr;
      oldSize, oldFont, oldMode : integer;
      oldStyle : Style;
      bmap, oldMap : BitMap;
  BEGIN
    GetPort(oldPort);
```

```
    SetPort(theMenuBar^^.gp);
    oldSize := thePort^.txSize;
    oldFont := thePort^.txFont;
    oldMode := thePort^.txMode;
    oldStyle := thePort^.txFace;
    TextSize(12);
    TextFont(systemFont);
    TextMode(srcOr);
    TextFace([]);

    SetRect(r, 0, 0, 10000, mBarHeight);
    EraseRect(r);
    MoveTo(0, mBarHeight - 1);
    Line(10000, 0);

    FOR i := 0 TO theMenuBar^^.numMenus - 1 DO
      BEGIN
        MoveTo(theMenuBar^^.wMenus[i].titleRect.left +
invertOverlap, mBarHeight - 5);
        DrawString(theMenuBar^^.wMenus[i].mh^^.menuData);

        { gray-out disabled menu titles }
        IF NOT
BitTst(@theMenuBar^^.wMenus[i].mh^^.enableFlags, menuTitleBit)
THEN
          BEGIN
            r := theMenuBar^^.wMenus[i].titleRect;
            r.left := r.left + invertOverlap;
            r.right := r.right - invertOverlap;
            bmap.bounds := r;
            OffsetRect(bmap.bounds, -r.left, -r.top);
            IF (bmap.bounds.right MOD 16) <> 0 THEN
              bmap.rowBytes := 2 * ((bmap.bounds.right DIV
16) + 1)
            ELSE
              bmap.rowBytes := 2 * (bmap.bounds.right DIV
16);
            bmap.baseAddr := NewPtr(bmap.rowBytes * mBar-
Height);
            oldMap := thePort^.portBits;
            SetPortBits(bmap);
            FillRect(bmap.bounds, gray);
            SetPortBits(oldMap);
            HLock(handle(theMenuBar));
            CopyBits(bmap, thePort^.portBits, bmap.bounds, r,
notSrcBic, NIL);
            HUnlock(handle(theMenuBar));
            DisposPtr(bmap.baseAddr);
          END; { if title disabled }
      END; { for each menu }
    IF theMenuBar^^.hilited <> noneHilited THEN

InvertRect(theMenuBar^^.wMenus[theMenuBar^^.hilited].titleRect);

    TextSize(oldSize);
    TextFont(oldFont);
    TextMode(oldMode);
    TextFace(oldStyle);
    SetPort(oldPort);
  END; { wDrawMenuBar }

{wDeleteMenu - delete a menu from a wMenuBar}
  PROCEDURE wDeleteMenu;   { (theMenuBar : wMenuBarHandle; }
                    { menuID : integer); }
    VAR
      i, j, oldSize, oldFont : integer;
      oldStyle : Style;
      titleWidth : integer;
  BEGIN
    oldSize := thePort^.txSize;
  { reset the font/size/style to compute menu title widths }
    oldFont := thePort^.txFont;
    oldStyle := thePort^.txFace;
    TextSize(12);
    TextFont(systemFont);
    TextFace([]);

    i := 0;
    WHILE (theMenuBar^^.wMenus[i].mh^^.menuID <> menuID)
    AND (i < theMenuBar^^.numMenus) DO
      i := i + 1;
    IF i <> theMenuBar^^.numMenus THEN
      BEGIN
        titleWidth :=
StringWidth(theMenuBar^^.wMenus[i].mh^^.menuData);
        FOR j := i TO theMenuBar^^.numMenus - 1 DO
          BEGIN
            theMenuBar^^.wMenus[j].mh :=
theMenuBar^^.wMenus[j + 1].mh;
            theMenuBar^^.wMenus[j].titleRect :=
theMenuBar^^.wMenus[j + 1].titleRect;
            OffsetRect(theMenuBar^^.wMenus[j].titleRect, -
(titleWidth + betweenTitles), 0);
          END; { for loop - copying menus back }
        theMenuBar^^.numMenus := theMenuBar^^.numMenus - 1;
      END; { if there's a menu id = menuID }

    TextSize(oldSize);
    TextFont(oldFont);
    TextFace(oldStyle);
  END; { wDeleteMenu }

{wClearMenuBar - delete all the menus in a menu bar}
  PROCEDURE wClearMenuBar; { (theMenuBar : wMenuBarHandle); }
  BEGIN
    theMenuBar^^.numMenus := 0; { take the easy way out.... }
  END; { wClearMenuBar }

{MenuDefProc - inline call to the menu definition procedure}
{   Pop the address of the proc off the stack and jsr to it}
  PROCEDURE MenuDefProc (message : integer;
                    theMenu : MenuHandle;
                    VAR menuRect : Rect;
                    hitPt : Point;
                    VAR whichItem : integer;
                    theProc : ProcPtr);
  INLINE
    $205F, $4E90; { pop.l A0 , jsr (A0) }

{MenuDefGlue - dereference menu handle to find the add. of}
{ the definition proc and call it using MenuDefProc, above}
  PROCEDURE MenuDefGlue (message : integer;
                    theMenu : MenuHandle;
                    VAR menuRect : Rect;
                    hitPt : Point;
                    VAR whichItem : integer);
  BEGIN
    MenuDefProc(message, theMenu, menuRect, hitPt, whichItem,
theMenu^^.menuProc^);
  END;

{wMenuSelect - pull down the menus and let the user select an
item}
  FUNCTION wMenuSelect;    { (theMenuBar : wMenuBarHandle; }
                      { startPt : Point) : longint; }
    CONST
      flashDelay = 3;      { # of ticks between calls to
invert selected item }
      menuFrame = 2;       { width of menu frame }
      MenuFlashAddr = $A24;
{ address of lo-mem global: # of times to flash menu }
    TYPE
      intPtr = ^integer;
    VAR
      bmap : BitMap;
      menuRect : Rect;
      oldClip : RgnHandle;
      nilPt : Point;
```

```
      blink, whichItem : integer;
      oldPort, wMgrPort : GrafPtr;
      i : integer;
      hstate, mprocState : SignedByte;
      ticks : longint;
      menuFlashP : intPtr;
      oldSize, oldFont, oldMode : integer;
      pnState : PenState;
      strayed : boolean;
      dummyEvt : EventRecord;
  BEGIN
    hState := HGetState(handle(theMenuBar));
    HLock(handle(theMenuBar));
    GetPort(oldPort);
    SetPort(theMenuBar^^.gp);
    oldSize := thePort^.txSize;
    oldFont := thePort^.txFont;
    oldMode := thePort^.txMode;
    TextSize(12);
    TextFont(systemFont);
    TextMode(srcOr);
    menuFlashP := intPtr(MenuFlashAddr);
    whichItem := 0;
    WHILE WaitMouseUp DO  { loop while the mouse is down }
      BEGIN
        { find menu title that user is clicking on }
        i := theMenuBar^^.numMenus - 1;
        WHILE (i >= 0) AND NOT PtInRect(startPt,
theMenuBar^^.wMenus[i].titleRect) DO
          i := i - 1;

{ if user is clicking menu title, have the menu "drop down" }
        IF i >= 0 THEN
          WITH theMenuBar^^.wMenus[i] DO
{ note: theMenuBar is locked }
            BEGIN
              wHiliteMenu(theMenuBar, mh^^.menuID);
{ hilite title }
              CalcMenuSize(mh);
{ calculate menu size, it may have changed }
              SetRect(menuRect, titleRect.left + 1, mBar-
Height, titleRect.left + mh^^.menuWidth + 1, mBarHeight +
mh^^.menuHeight);
              InsetRect(menuRect, -menuFrame, -menuFrame);


{ if the menu overlaps the edges of the window, trim it }
          IF menuRect.bottom > thePort^.portRect.bottom THEN
              menuRect.bottom := thePort^.portRect.bottom;
            IF menuRect.right > thePort^.portRect.right THEN
              OffsetRect(menuRect, thePort^.portRect.right
- menuRect.right - 2, 0);
            IF menuRect.left < 0 THEN
              OffsetRect(menuRect, -menuRect.left, 0);
            bmap.rowBytes := ((menuRect.right -
menuRect.left + 15) DIV 16) * 2;
            bmap.bounds := menuRect;
            bmap.baseAddr := NewPtr(bmap.rowBytes *
(menuRect.bottom - menuRect.top));
            IF bmap.baseAddr <> NIL THEN
            { proceed if there is memory }
              BEGIN
                CopyBits(thePort^.portBits, bmap,
bmap.bounds, bmap.bounds, srcCopy, NIL);
                oldClip := NewRgn;
                GetClip(oldClip);
                ClipRect(menuRect);


{ draw the menu - thanks to Mike Schuster, MacTutor 12/85 }
                IF mh^^.menuHeight > 0 THEN
                  BEGIN
                    InsetRect(menuRect, menuFrame,
menuFrame);
                    EraseRect(menuRect);
                    InsetRect(menuRect, -1, -1);
                    FrameRect(menuRect);
                    InsetRect(menuRect, 1, 1);

                    GetPenState(pnState);
                    PenNormal;
                    MoveTo(menuRect.left + 1,
menuRect.bottom + 1);
                    Line((menuRect.right -
menuRect.left), 0);
                    Line(0, -(menuRect.bottom -
menuRect.top));
                    SetPenState(pnState);
                  END; { if there're any menu items }

                LoadResource(mh^^.menuProc);
                mprocState :=
HGetState(handle(mh^^.menuProc));
                HLock(mh^^.menuProc);
                whichItem := 0;
                MenuDefGlue(mDrawMsg, mh, menuRect,
startPt, whichItem);

{ send the mChooseMsg while the user is still in this menu }
                strayed := false;
                WHILE WaitMouseUp AND NOT strayed DO
                  BEGIN
                    MenuDefGlue(mChooseMsg, mh,
menuRect, startPt, whichItem);
                    GetMouse(startPt);
                    strayed := (startPt.v < mBarHeight
- 1) AND (startPt.v > 0) AND (NOT PtInRect(startPt, ti-
tleRect));
                  IF i < theMenuBar^^.numMenus - 1 THEN
                    strayed := strayed OR ((startPt.v <
mBarHeight - 1) AND (startPt.v > 0) AND PtInRect(startPt,
theMenuBar^^.wMenus[i + 1].titleRect));

{ Enable FKeys (i.e. screen dump) and DA updating }
                    IF EventAvail(everyEvent, dummyEvt)
THEN
                      ;
                    SystemTask;
                  END; { while WaitMouseUp &not strayed }

                { flash the menu if an item was selected }
                IF (whichItem <> 0) AND NOT strayed THEN
                  FOR blink := 1 TO menuFlashP^ DO
                    BEGIN
                      SetPt(nilPt, 0, 0);
                      MenuDefGlue(mChooseMsg, mh,
menuRect, nilPt, whichItem);
                      Delay(flashDelay, ticks);
                      MenuDefGlue(mChooseMsg, mh,
menuRect, startPt, whichItem);
                      Delay(flashDelay, ticks);
                    END; { whichItem <> 0 }
                HSetState(mh^^.menuProc, mprocState);
                SetClip(oldClip);
                DisposeRgn(oldClip);
                CopyBits(bmap, thePort^.portBits,
bmap.bounds, bmap.bounds, srcCopy, NIL);
                DisposPtr(bmap.baseAddr);
              END; { memory for bitmap }
          END { i >= 0: found the hit menu title }
        ELSE
          BEGIN
{ user isn't over a menu, so unhilite the last one hilited }
            IF theMenuBar^^.hilited <> noneHilited THEN
              wHiliteMenu(theMenuBar, 0);
            GetMouse(startPt);
{ need a new startPt - mouse may have moved }
          END;{ no menu currently selected }
```

```
      END; { while WaitMouseUp - looking for a hit menu title}

  { user let up on the mouse - return the appropriate value }
    IF whichItem = 0 THEN
      wMenuSelect := 0
    ELSE
      wMenuSelect :=
BitShift(theMenuBar^^.wMenus[i].mh^^.menuID, 16) + whichItem;
    TextSize(oldSize);
    TextFont(oldFont);
    TextMode(oldMode);
    SetPort(oldPort);
    HSetState(handle(theMenuBar), hState);
  END; { wMenuSelect }

{wMenuKey - return the menu id and item no. with ch as it's}
{ cmd-key equivalent }
{ Caution: this assumes knowledge of the internal structure}
{ of  MenuInfo.menuData }
  FUNCTION wMenuKey;       { (theMenuBar : wMenuBarHandle; }
                      { ch : char) : longint; }
    CONST
      flashDelay = 3;
    TYPE
      SignedBytePtr = ^SignedByte;
      CharPtr = ^char;
    VAR
      hState : SignedByte;
      bp, keyEquivP : SignedBytePtr;
      i, j, whichMenu, whichItem : integer;
      done, enabled : boolean;
      ticks : longint;

  { compare alphabetic characters w/o case sensitivity }
    FUNCTION equalChars (c1, c2 : char) : boolean;
    BEGIN
      IF c1 IN ['a'..'z'] THEN
        c1 := char(ord(c1) + ord('A') - ord('a'));
      IF c2 IN ['a'..'z'] THEN
        c2 := char(ord(c2) + ord('A') - ord('a'));
      equalChars := c1 = c2;
    END;

  BEGIN
    i := 0;
    done := false;
    WHILE (NOT done) AND (i < theMenuBar^^.numMenus) DO
      BEGIN
        hState :=
HGetState(handle(theMenuBar^^.wMenus[i].mh));
        HLock(handle(theMenuBar^^.wMenus[i].mh));

{ run down a menu, looking for an item w/ ch as its key}
{ equivalent }
        j := 1;
        bp :=
SignedBytePtr(@theMenuBar^^.wMenus[i].mh^^.menuData);
        bp := SignedBytePtr(ord4(bp) + bp^ + 1);
        WHILE (NOT done) AND (bp^ <> 0) DO
          BEGIN
            keyEquivP := SignedBytePtr(ord4(bp) + bp^ + 2);
            IF equalChars(ch, char(keyEquivP^)) THEN
              BEGIN
                whichMenu :=
theMenuBar^^.wMenus[i].mh^^.menuID;
                whichItem := j;
                done := true;
              END
            ELSE
              BEGIN
                j := j + 1;
                bp := SignedBytePtr(ord4(keyEquivP) + 3);
              END;
          END; { looking through this menu }

        HSetState(handle(theMenuBar^^.wMenus[i].mh), hState);
        i := i + 1;
      END; { while loop - looking for key equiv }
    {the item is enabled if both it and its menu title are}
    enabled := BitTst(@theMenuBar^^.wMenus[i -
1].mh^^.enableFlags, menuTitleBit);
    enabled := enabled AND (j < 32) AND
(BitTst(@theMenuBar^^.wMenus[i - 1].mh^^.enableFlags, menuTi-
tleBit - j));
    IF done AND enabled THEN
      BEGIN
        wHiliteMenu(theMenuBar, whichMenu);
        wMenuKey := BitShift(whichMenu, 16) + whichItem;
      END { done }
    ELSE
      wMenuKey := 0;
  END; { wMenuKey }

{wHiliteMenu - unhilite the currently hilited menu title, and}
{ hilite a new one }
{ if menuID <> 0 }
  PROCEDURE wHiliteMenu;   { (theMenuBar : wMenuBarHandle; }
                      { menuID : integer); }
    VAR
      i : integer;
      oldPort : GrafPtr;
  BEGIN
    GetPort(oldPort);
    SetPort(theMenuBar^^.gp);
    IF (theMenuBar^^.hilited <> noneHilited) THEN

InvertRect(theMenuBar^^.wMenus[theMenuBar^^.hilited].titleRect);
    theMenuBar^^.hilited := noneHilited;
    IF menuID <> 0 THEN
      BEGIN
        i := 0;
        WHILE (theMenuBar^^.wMenus[i].mh^^.menuID <> menuID)
AND (i < theMenuBar^^.numMenus) DO
          i := i + 1;
        IF i <> theMenuBar^^.numMenus THEN
          BEGIN
            InvertRect(theMenuBar^^.wMenus[i].titleRect);
            theMenuBar^^.hilited := i;
          END; { found menuID }
      END;  { menuID <> 0 }
    SetPort(oldPort);
  END; { wHiliteMenu }

END.
```

```
{wSample - the Mac User Education prog, adapted to use wMenus}
{ by Jim Matthews }
PROGRAM wSample;
{$I-}
  USES
    wMenu;
  CONST
    appleID = 128;
    fileID = 129;
    editID = 130;

    appleM = 1;
    fileM = 2;
    editM = 3;

    menuCount = 3;

    windowID = 128;

    undoCommand = 1;
    cutCommand = 3;
    copyCommand = 4;
    pasteCommand = 5;
```

```
    clearCommand = 6;

  VAR
    myMenus : ARRAY[1..menuCount] OF MenuHandle;
    dragRect, txRect : Rect;
    textH : TEHandle;
    theChar : char;
    extended, doneFlag : boolean;
    myEvent : EventRecord;
    wRecord : WindowRecord;
    myWindow : WindowPtr;
    whichWindow : WindowPtr;
    myMenuBar : wMenuBarHandle;

{SetUpWMenus - read in menu templates and set up wMenuBar}
  PROCEDURE SetUpWMenus;
    VAR
      i : integer;
  BEGIN
    myMenuBar := wInitMenus(myWindow);
    myMenus[appleM] := GetMenu(appleID);
    AddResMenu(myMenus[appleM], 'DRVR');
    myMenus[fileM] := GetMenu(fileID);
    myMenus[editM] := GetMenu(editID);

    FOR i := 1 TO menuCount DO
      wInsertMenu(myMenuBar, myMenus[i], 0);
  END; {SetUpWMenus}

{DoCommand - handle menu commands}
  PROCEDURE DoCommand (mResult : longint);
    VAR
      theItem, theMenu : integer;
      name : Str255;
      temp : integer;
  BEGIN
    theItem := LoWord(mResult);
    theMenu := HiWord(mResult);

    CASE theMenu OF
      appleID :
        BEGIN
          GetItem(myMenus[appleM], theItem, name);
          temp := OpenDeskAcc(name);
          SetPort(myWindow);
        END; { appleID }
      fileID :
        doneflag := true;
      editID :
        BEGIN
          IF NOT SystemEdit(theItem - 1) THEN
            CASE theItem OF
              cutCommand :
                TECut(textH);
              copyCommand :
                TECopy(textH);
              pasteCommand :
                TEPaste(textH);
              clearCommand :
                TEDelete(textH);
            END; { case theItem of }
        END; { editID}
      OTHERWISE
        ;
    END; { case theMenu of }
    wHiliteMenu(myMenuBar, 0);
  END; { DoCommand }

{main program}
BEGIN
  InitGraf(@thePort);
  InitFonts;
  FlushEvents(everyEvent, 0);
  InitWindows;
  InitMenus;
  TEInit;
  InitDialogs(NIL);
  InitCursor;

  WITH screenBits.bounds DO
    SetRect(dragRect, 4, 24, right - 4, bottom - 4);
  doneFlag := false;
  myWindow := GetNewWindow(windowID, @wRecord, WindowPtr(-1));
  SetPort(myWindow);
  SetUpWMenus;
  txRect := thePort^.portRect;
  txRect.top := mBarHeight;
  InsetRect(txRect, 3, 3);
  textH := TENew(txRect, txRect);

  {main event loop}
  REPEAT
    SystemTask;
    TEIdle(textH);

    IF GetNextEvent(everyEvent, myEvent) THEN
      CASE myEvent.what OF
        mouseDown :
          CASE FindWindow(myEvent.where, whichWindow) OF
            inSysWindow :
              SystemClick(myEvent, whichWindow);
            inMenuBar :
              IF MenuSelect(myEvent.where) <> 0 THEN
                ;
{ handle da menus in the "real" menu bar }
            inDrag :
              DragWindow(whichWindow, myEvent.where, dra-
gRect);
            inContent :
              BEGIN
                GlobalToLocal(myEvent.where);
                IF myEvent.where.v < mBarHeight THEN
                  BEGIN
                    IF whichWindow <> FrontWindow THEN
                        SelectWindow(whichWindow);
                    DoCommand(wMenuSelect(myMenuBar,
myEvent.where));
                  END
                ELSE
                  BEGIN
                    IF whichWindow <> FrontWindow THEN
                        SelectWindow(whichWindow)
                    ELSE
                        BEGIN
                            extended :=
BitAnd(myEvent.modifiers, shiftKey) <> 0;
                            TEClick(myEvent.where,
extended, textH);
                        END; { whichWindow = FrontWindow}
                  END; { click below menu bar }
              END; { inContent }
            OTHERWISE
              ;
          END; { mouseDown }
        keyDown, autoKey :
          BEGIN
            theChar := char(BitAnd(myEvent.message, charCode-
Mask));
            IF BitAnd(myEvent.modifiers, cmdKey) <> 0 THEN
              DoCommand(wMenuKey(myMenuBar, theChar))
            ELSE
              TEKey(theChar, textH);
          END; { keyDown, autoKey }
        activateEvt :
          BEGIN
            IF BitAnd(myEvent.modifiers, activeFlag) <> 0
THEN
              BEGIN
```


© The Definitive MacTutor, Vol. 4

```
            TEActivate(textH);
            DisableItem(myMenus[editM], undoCommand);
          END
        ELSE
          BEGIN
            TEDeactivate(textH);
            EnableItem(myMenus[editM], undoCommand);
          END; { activate/deactivate }
      END; {activateEvt }
    updateEvt :
      BEGIN
        BeginUpdate(WindowPtr(myEvent.message));
        EraseRect(thePort^.portRect);
        wDrawMenuBar(myMenuBar);
        TEUpdate(thePort^.portRect, textH);
        EndUpdate(WindowPtr(myEvent.message));
      END; { updateEvt }
    OTHERWISE
      ;
  END; { case event.what of }
UNTIL doneFlag;
END.
```

```
* Rmaker source for wSample.rsrc

wSample.rsrc
????????
```

```
TYPE WIND
   ,128
wSample
40 50 300 450
Visible NoGoAway
0
0

TYPE MENU
   ,128
\14


TYPE MENU
   ,129
File
  Quit/Q

TYPE MENU
   ,130
Edit
  (Undo/Z
  (-
  Cut/X
  Copy/C
  Paste/V
  Clear
```

Pascal Procedures

Inside Macinkeys

Fabien Samuel
Paris France
MacTutor Vol. 4 No. 12

Fabien Samuel is a mathematics teacher in Paris, as well as a convinced Macintosh adept and Programmer.

Getting Info

This article is meant to give a comprehensive overview about the various resources, structures and procedures involved in keyboard events, and especially about the relation between Keycodes and Ascii Codes. There has already been a couple of very good articles on that subject in MacTutor, and I must thank Joel West and Jörg Langowski for the help I found in these as a starting point. The present article tries to update and complete the information with respect to the new resources in System 4.0 or later.

As an example, I have included the KeyEdDemo application, which allows you to remap your keyboard according to specific needs and save that configuration both in RAM, for immediate access, and in your system file, so as to keep the changes after reboot. This is a very simplified version of my shareware application KeyEd 0.5, that features a fully functional file menu, multiple scrolling windows, extensive font and view options, and advanced error checking. The main purpose of this demo version is to show how the resources described below can be used.

The information presented here has been obtained by various means. Lacking documentation about the new resources introduced with system 4.0, the most valuable help was found in MacNosy. I had quite the feeling of being a detective, looking through the nosied code of INIT 0 and _KeyTrans to deduce the structure of KCHR resources.

When I finally got Inside Mac Vol V, I was able to confirm some of my discoveries and find some more information, although the information presented there is not always very reliable.

Transitions and Codes

The keyboard contains its own micro-processor, which returns to the Mac OS Transitions Codes, also known as Raw Key Codes, every time the user types a key. Those Raw Key Codes are then translated in Virtual Key Codes, involving the use of a translation table contained in KMAP resources. The details of this translation are beyond the scope of this article, and in everything that follows we will call Key Codes the virtual key codes that were so obtained.

Key Codes can take any value between 0 and 127, not all values being actually used (at least until they release a super-extended keyboard with 128 keys!). A 16 bytes long variable, found in the low-memory global KeyMap ($174), tells which keys are down at any given time: each set bit indicating that the corresponding key is down. At least 2 normal keys can be thus indicated, along with any combination of the modifiers keys (Shift, option, etc...).

The contents of this variable can also be obtained through the Pascal procedure GetKeys (TheKeys: KeyMap). A few important notes must be made here. First of all, although the standard interface for Type KeyMap is Packed Array [0..127] of Boolean, a limitation of the LightSpeed Pascal compiler redefines it as Array [0..3] of LongInt. The underlying structure remains the same, but it cannot be accessed in the same manner. Second, as shown in Figure 1, KeyCodes do not directly correspond to a bit offset in the KeyMap variable, the order of bits being reversed in each byte. This can be very convenient at the assembly language level, where you just need to shift the KeyCode by 3 bits to get the byte offset, the remainder being the bit offset within that byte that can be used along with instructions like BTST.

Byte :	bit 0	bit 1	bit 2	bit3	bit 4	bit 5	bit6	bit 7	Byte :	bit 0	bit 1	bit 2	bit3	bit 4	bit 5	bit6	bit 7
$174	A $0	S $1	D $2	F $3	H $4	G $5	Z $6	H $7	$175	C $8	U $9	$A	B $B	Q $C	W $D	E $E	R $F
$176	Y $10	T $11	1 $12	2 $13	3 $14	4 $15	6 $16	5 $17	$177	= $18	9 $19	7 $1A	- $1B	8 $1C	0 $1D	] $1E	O $1F
$178	U $20	[$21	I $22	P $23	return $24	L $25	J $26	' $27	$179	K $28	; $29	\ $2A	, $2B	/ $2C	N $2D	M $2E	. $2F
$17A	tab $30	$31	$32	delete $33	$34	esc $35	$36	⌘ $37	$17B	shift $38	caps lock $39	option $3A	control $3B	r shift $3C	r option $3D	r control $3E	$3F
$17C	$40	. $41	$42	* $43	$44	+ $45	$46	clear $47	$17D	$48	$49	$4A	/ $4B	enter $4C	$4D	- $4E	$4F
$17E	$50	= $51	0 $52	1 $53	2 $54	3 $55	4 $56	5 $57	$17F	6 $58	7 $59	$5A	8 $5B	9 $5C	$5D	$5E	$5F
$180	F5 $60	F6 $61	F7 $62	F3 $63	F8 $64	F9 $65	$66	F11 $67	$181	$68	F13 $69	$6A	F14 $6B	$6C	F10 $6D	$6E	F12 $6F
$182	$70	F15 $71	help $72	home $73	page up $74	del ⌦ $75	F4 $76	end $77	$183	F2 $78	page down $79	F1 $7A	← $7B	→ $7C	↓ $7D	↑ $7E	◁ $7F
Word:	bit 8	bit 9	bit 10	bit 11	bit 12	bit 13	bit 14	bit 15		bit 0	bit 1	bit 2	bit3	bit 4	bit 5	bit6	bit 7

Figure 1
Relation between KeyMap and Key Codes
KeyCaps from the Extended US ADB Keyboard have been shown along with the hex key codes , and the bit offset information .

© The Definitive MacTutor, Vol. 4

Things get more complicated in Pascal if one wants to use the bit-wise ToolBox utilities, as bits are computed in the reverse order (Higher to Lower bits, Lower to Higher Bytes) than they are in the corresponding assembly instructions. However the following code shows a way to access a KeyMap that will work with any compiler:

```
Var TheKeys:KeyMap; KeyCode,Offset:Integer;
GetKeys(TheKeys);
Offset:= KeyCode + 7 - 2 * (KeyCode mod 8);
if BitTst(@TheKeys,Offset) then { Key is down }
```

Different KeyBoards

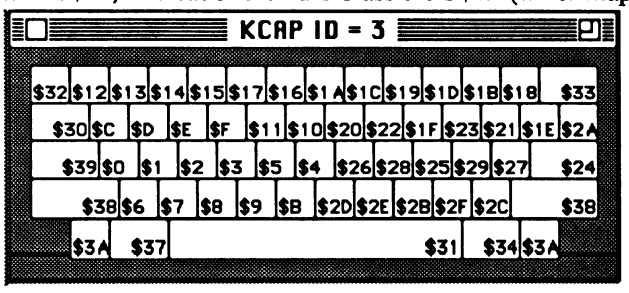
According to the countries and Mac models, the relation between physical keys and their key codes can vary greatly. This can be due to distinct physical mappings, for instance the return key or the cursor keys can be found at different locations from one keyboard to another. To make things even simpler, keys having the same function can have different key codes, or, better still, keys having the same keycodes can have different meanings! We will try to summarize these exceptions in a moment.

There currently are 6 different Apple keyboards, each one having its own keyboard type (Found in low-memory global KybdType = $21E - Byte value). However the keyboard types for the US classic and Euro classic keyboards are both equal to 3, despite their important differences: a way to distinguish between them is shown in the demo program (Function GetKbd) . A display map of these keyboards can be found in KCAP resources, contained in the system file "Key Layout", that we will discuss later.

In chronological order, the Mac Classic (KybdType = 3) keyboards exist in US Version (KCAP ID = 3) and International Version (KCAP ID = 259). For archaeologists that might be interested, those are the keyboards that were released with the antiques 128 and 512 Macs. Then came the Mac Plus keyboard (KybdType = KCAP ID = 11), and the 3 ADB Keyboards released with the SE and Mac II: Standard (ID = 1), Extended (ID = 2) and ISO (ID = 4), this last one satisfying with the international ISO specifications, unlike the Standard ADB Keyboard. You can find representations of all those keyboards in Figures 2 to 7, along with the corresponding key codes.

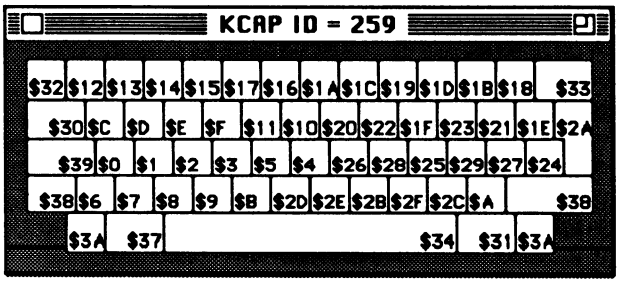
We can now see better what are the main differences between keyboards:

1. Return and \: On the US Classic Keyboard the keycode for Return is $24, whereas on the Euro Classic it is $2A (which maps



KCAP ID = 3 : US CLASSIC KEYBOARD

Figure 2



KCAP ID = 259 : EURO CLASSIC KEYBOARD

Figure 3

to the \ character on the US Classic). In all other keyboards, the keycode for Return is still $24, and in all cases the ascii code is and should be $0D.

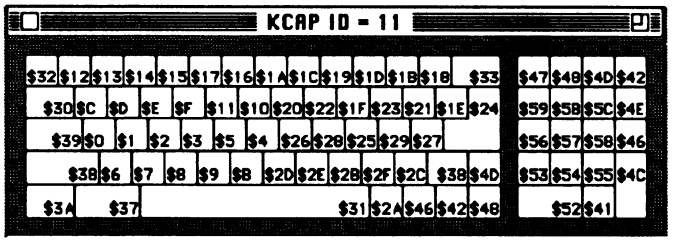
2. Enter and Space: again the corresponding keycodes ($34 for Enter and $31 for Space on the MacClassic) have been inverted on the EuroClassic keyboard. On all subsequent keyboards, the Enter key has been moved to the numeric keypad, with keycode $4C, the Space key keeping its $31 keycode.

3. Cursor Keys: originally, those keys were only available on the optional numeric keypad, where they also served as operator keys when shifted. With the introduction of the Mac Plus built-in numeric keypad, the cursor keys were moved to the main keyboard, while the numeric operator keys stayed where they belonged on the keypad. The only problem is that they kept the same keycodes! To solve this conflict, when you type one of the numeric operator keys on the Plus, it acts as if you had held the shift key down (and likewise, holding down the shift key while typing one of the cursor keys will return the corresponding numeric operator). On subsequent models, this conflict was resolved by giving altogether different keycodes to both the numeric operator and cursor keys: this means 8 new key codes have been introduced for that purpose. The situation is summarized in Figure 8. Also note that the + and - keys on the keypad have been inverted between the ADB extended and standard keyboards.

The main conclusion of all this is that you should only rely on the ascii code of such control characters, which always keeps the same value whatever keyboard you use.

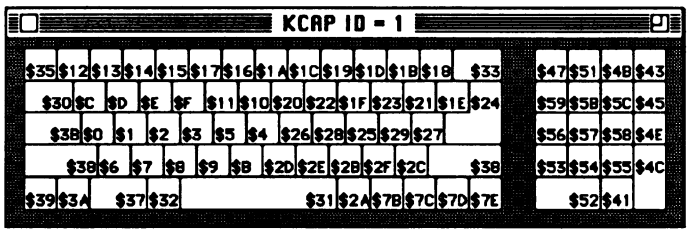
Keyboard Events

For most purposes, dealing with keyboard input mainly involves polling KeyDown and AutoKey events. Information about such an event is given in the Message and Modifiers fields



KCAP ID = 11 : MAC PLUS KEYBOARD

Figure 4



KCAP ID = 1 : ADB STANDARD KEYBOARD

Figure 5

of the EventRecord returned by GetNextEvent. The Message field has the following 4 bytes long structure:

<Reserved, ADBAddress, keycode, ascii code>

ADBAddress is used with the Mac SE and II, and contains a reference number of the keyboard the typing came from, useful if various keyboards are simultaneously connected (see January 88 MacTutor for an example of this). Ascii code is obtained through the conversion routines that are discussed in the next paragraphs, taking into account both the key code and the state of modifiers keys.

The Modifiers field of the EventRecord is made up of 2 bytes, the high byte giving the state of the modifiers keys in the following bit-arrangement:

<x,y,z,control,option,caps lock,shift,command>

Flags x,y and z are normally zero. However, the extended ADB keyboard can be reconfigured so as to distinguish between the left and right Shift,Option and Control Keys. In that case, the right keys will generate new key codes, and flags x,y,z could be renamed: R.Control,R.Option,R.Shift. This possibility is supported in order to use the Mac with an operating system Inside Mac doesn't dare to mention, so neither will I! It can be implemented through the use of the ADB operators mechanism, which might be the subject of another article. Anyway, this capacity is strongly discouraged because it doesn't comply with the User Interface.

Finally, let's note that some keys do not immediately return an event: those are the keys that are used in conjunction with another key to produce exotic objects like accented characters. I will refer to them as "Double Strike Keys", also the official terminology appears to be "Dead Keys", but I just don't like the idea of my keys being dead!

In the beginning, there was INIT 0 and 1...

On the first Macs, and until the Mac Plus with systems 3.x, all the information related to converting key codes to ascii codes was contained in system resources INIT 0 and 1. The Localizer was in charge of adapting these resources according to the various countries. Those resources contained both the conversion code and the associated translation tables.

With systems 4.x and later, The INIT's now mainly contain code, the translation tables being now separate and contained in the KCHR 0 system resource. At boot time, the INIT's are loaded into the system heap and jumped to. They are mainly in charge of installing pointers to the conversions routines in low-memory globals Key1Trans and Key2Trans.

The reason for there being two separate routines is due to the fact that originally, the numeric keypad was optional. Key1Trans will thus deal with keycodes from 0 to 63, while Key2Trans treats keycodes from 64 to 127, that used to be found exclusively on the numeric keypad. This distinction now appears completely artificial (if not intelligent...) since the Mac Plus was introduced. It seems that those routines have only stayed for compatibility reasons, as their codes are now almost the same, and both refer to the new _KeyTrans ToolBox trap, which does most of the work.

The Inner Workings

To start with, I will describe the main steps involved in INIT 0, INIT 1 having a similar functionality. As any INIT resource, it gets executed at boot time after being loaded as a locked relocatable block, accessed through a handle. It's made up of two essential routines that I'll call SetUp and Key1TransMain. SetUp is the part that gets jumped to at boot time and does the following :

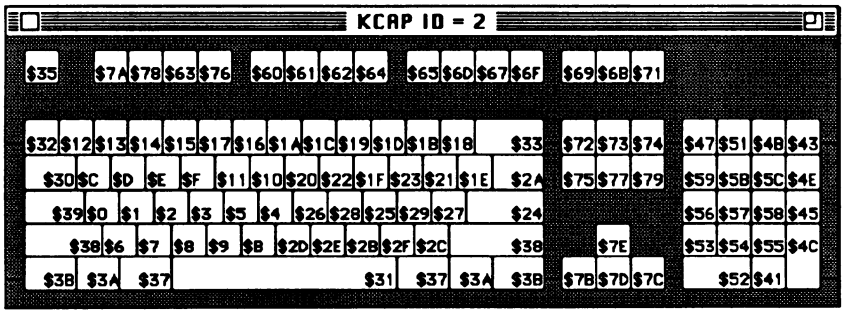
1. Load KCHR 0 into the system heap, if it's not already there.

2. Create a non-relocatable copy of KCHR 0 in system heap and store a pointer to it at offset 14 in a parameter block that is accessed through the pointer maintained in BasicGlob (= $2B6) low-memory global. Actually, this copy is expanded by at least 512 bytes, probably in order to allow its modification by another program.

3. Copy Byte 6 from the itlc 0 in a Data zone internal to INIT 0. This is an international utilities resource, and the value of this byte will be used to distinguish between the US Classic and Euro Classic keyboards, as illustrated in the GetKbd function of the demo program. This actually must mean that running with an Euro Classic keyboard and a US system would make believe the keyboard was US *[This is true, recalling all the mixups we had using different systems - JL]*: there doesn't seem to be a hardware method to distinguish between the two keyboards.

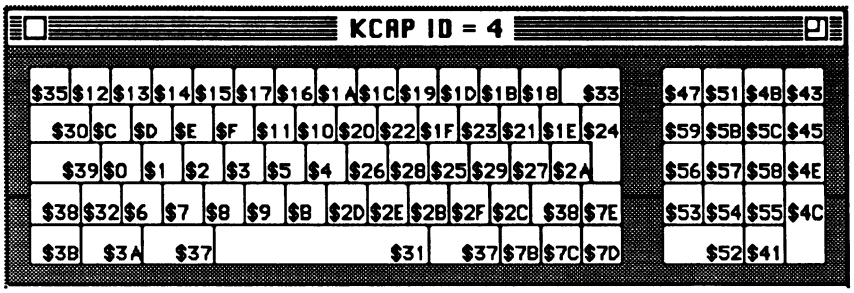
4. Put the entry point to Key1TransMain in low-memory global Key1Trans.

5. Cut down the size of its own handle before returning, the SetUp routine being now useless (this is accomplished by putting those last instruc-



KCAP ID = 2 : ADB EXTENDED KEYBOARD

Figure 6



KCAP ID = 4 : ADB ISO KEYBOARD

Figure 7

tions at the beginning of the block, unless one wants to call this the Auto-Destroy routine!).

Key1Trans

Key1transMain is the part currently used every time a keydown or keyup event is detected by the Event Manager. This is a register-based routine, and calling it from an high level language involves assembler glue-code (see the Demo Program). It takes on entry:

D2 contains the keycode (less than 64, or nothing will happen: in that case one must call Key2Trans)

D1 contains the state of the modifiers keys (as it appears in the third word of the KeyMap structure: this differs from the EventRecord Modifiers field).

D3 should be 0 for most purposes: bit 7 can be set to indicate that the event should be treated as a key-up event rather than a key-down event.

On exit, D0 contains the resulting ascii code.

This routine works as follows:

1. Make up a word that has the following structure:

- the high byte contains the modifier flags as they would appear in the EventRecord modifiers field:

<x,y,z,ctl,opt,caps,shift,cmd>

Note that the command key state is read directly on the keyboard when the routine is executed, and so doesn't depend on the one you pass in D1.

- the low byte contains the key code, with a flag in bit 7 indicating KeyUp status.

Key	Ascii	Plus Code	ADB Code	Key	Ascii	Plus Code	ADB Code
°	$1C	$46	$7B	*	$2A	$46	$43
·	$1D	$42	$7C	+	$2B	$42	$45
·	$1E	$4D	$7E	/	$2F	$4D	$4B
-	$1F	$48	$7D	=	$3D	$48	$51

Figure 8

2. Call the ToolBox function

KeyTrans (transData:Ptr; KeyCode:Integer; VAR State: LongInt): Longint;

where KeyCode is the word obtained in step 1, TransData a pointer to the copy of KCHR 0 accessed through BasicGlob as described above, and State an offset used for double strike keys that is maintained at offset 10 in the parameter block pointed to by BasicGlob. We will see how this works in just a moment .

The function result contains in its Low Word the ascii code of the current key, taking into account its possible combination with a previous double strike key. The High Word returns the ascii code for an eventually waiting double strike if it couldn't be combined with the current key.

3. In case the High Word is non-zero, Key1Trans will post an event for it, thus causing the default ascii value of the previous double strike to be reported before that of the current key.

4. In any case, the ascii code for the current key and modifiers is returned in D0.

KeyTrans and KCHR 0

What finally remains to complete this overview is to understand how the KeyTrans function is working. In order to do that, we first need to know the structure of the KCHR 0 resource, which will incidentally give us the means to modify it.

As I said, KCHR 0 is the fundamental resource that allows to translate a keycode, along with the associated modifiers, to a standard ascii code.

1. The first word of a KCHR resource presently contains zero, and doesn't seem to be currently used.

2. The next 256 bytes serve as a kind of offset table, each one containing a block number used as follows: the high byte of the Event Record modifiers field can theoretically take any value between 0 and 255, assuming the right modifiers keys are activated. This value serves as an offset into the table to get the corresponding block number. Each block is 128 bytes long, and contains the mapping for every key code when typed along with the above modifiers.

3. The following word contains the actual number of blocks that are used (currently 8, numbered from 0 to 7). This means different modifiers combinations can lead to the same block.

4. Then come the block themselves, each of its 128 bytes giving the ascii code for the corresponding key code, which serves as an offset into the block. If the ascii code is zero, it means it's either a double strike key, a modifier key, or no key at all.

5. Finally, we find a variable length part that deals with double strikes.

a) The first word contains the total number of double strike keys.

b) For each one of them, we find a structure organized as follows:

b1) 2 bytes, giving the block number and key code of that double strike.

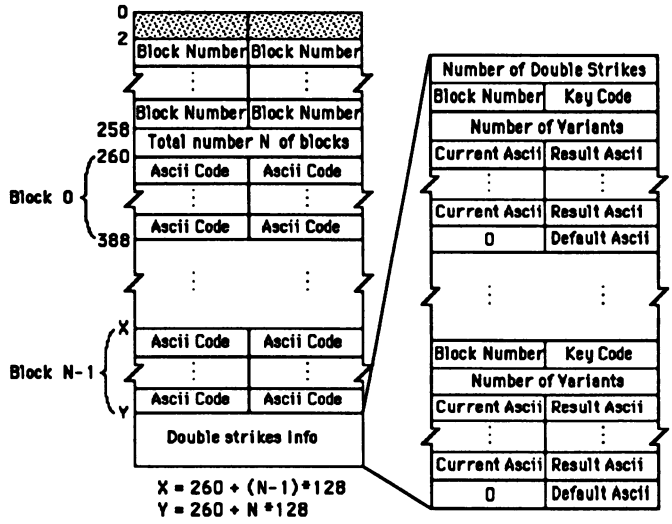


Figure 9

b2) Word giving the total number of associated variants.

b3) List of all those variants in the form: Current Ascii Code / Resulting Ascii Code

b4) Word giving the default ascii value of that double strike, in case the current ascii code wouldn't combine with it to produce a resulting ascii code.

Let's give an example to try and make things a bit clearer: on the US keyboard, **Option E** followed by **E** results in the accented Character **é**, but if followed by **P** it will result in the two consecutive characters **´p**. The keycode for **E** is $E, while the option key gives a value of 8 to the modifiers flags high byte. Counting from zero, byte number 8 in the block numbers table has a value of 3, which leads us to block number 3. Looking at byte number $E in that block, we see that its value is zero! This confirms us in thinking that **Option E** is a double strike. Now, let's look at the list of double strike keys: the first of them is precisely the one we want: Block Number = 3, KeyCode = $E. The next word tells us 7 variants are associated with it, and among them we find the variant $65,$8F: ascii code $65 (Character **e**) thus combines with our double strike to produce ascii code $8F (Character **é**). On the other hand, no variant is associated with ascii code $70 (Character **p**), and it will thus be the default value $60 (´ accent) that will be returned, followed by the character **p**.

We finally can explain how the KeyTrans function does its trick. The Modifiers byte is used as an offset into the block numbers table, while the keycode byte serves as an offset into the designated block. If the ascii code it finds at the designated position is non-zero, KeyTrans just returns with it. If it is zero, KeyTrans looks into the double-strike table and tries to match the BlockNumber/Keycode combination with one in the list. If it succeeds, the State variable is loaded with the byte offset between the start of the first block and the location of the "number of variants" entry associated with that double strike. In all other cases, State will be cleared. When KeyTrans is called again, and State is non-zero, it assumes there's a pending double strike and, after getting the current ascii code, it will look right away at the designated offset to find a match between the current ascii code and an eventual resulting ascii code. If it finds one, it returns with the resulting ascii code. Otherwise it returns with the current ascii code in its low word, and the default ascii code of the pending double strike in its high word. In the last case, and if it has been called by KeyITrans, KeyITrans will post the appropriate event as described above. We can thus see how important it is to maintain the value of State between calls to KeyTrans if we want to get double strikes responses.

As a final point, note that KeyTrans actually does some pre-processing related to the Script Manager.

The Demo Program

The following LightSpeed Pascal program tries to give a simple example of how all this stuff can be put to use. It needs system 4.0 or later to run. It only allows editing of normal keys, and I left out all memory and resources error checking in order to keep it as short as possible. Note that the Shareware version KeyEd 0.5 does implement advanced error-checking features. I'll leave it as an exercise to the reader to check for Nil Handles and call MemError or ResError when appropriate. However, I couldn't resist showing how the KCAP's resources mentioned above can be used in giving a graphical interface to keyboard editing, and that accounts for a large part of the program code. So I first have to say a word about their structure (obtained once again by nosying the KeyCaps DA):

1. Window Rectangle
2. KeyCaps DA TextEdit view rectangle
3. Number of different key shapes
4. For each shape:
   a) Number of points used to define it, minus 1.
   b) List of those points.
   c) Number of associated keys, minus 1.
   d) List of those keys in the form:
   <KeyCode,Vertical Offset, Horizontal Offset>.

The points define a region made up of rectangles, that will then be offset to the adequate position for each consecutive key. Note that the keycode field actually contains additional information, so I take it modulo 128. The procedure MainCaps that runs through this structure shows you type-casting in Pascal can do wonders, but would evidently be more elegant in assembly! The only thing I wasn't able to do with type-casting was to access a byte at a time in memory, so I wrote the short assembly routines Peek and Poke for that purpose, the names being a tribute to Basic programmers.

The program also shows an example of using procedural parameters, a most convenient feature of the LSP compiler. This way the region obtained in MainCaps can be used either for drawing the key or for dealing with a click inside it, or for any other purpose you might want. However, if you want to port that program to another compiler, you'll need to implement a Case statement within MainCaps to achieve the same result.

Finally, the program also shows the use of a modeless dialog with an User Item. The user procedure UserDraw is attached to that item with the SetUserProc procedure. The Dialog Manager

will call it automatically on each update event for the dialog window, and it is in charge of drawing the keyboard. Likewise, if there was a click in that User Item rectangle, we call MainCaps to find if it was in a key, and handle the event accordingly in the ActiveClick procedure. Note that although the Dialog Manager deals with mouse-down events when the Dialog Window is active, you still have to explicitly activate it by calling Select Window when the Event Manager reports a click in the contents region. You also have to pre-process keydown events, to filter out command key equivalents and any other typing you don't want passed to the edit field. One important warning if you use LightsBug while running this program is that if you inadvertently click in your switched-out Dialog Window, LSP 1.11 will bomb after displaying the usual "You can only drag your program's windows while it is halted" message. This could be due to the fact that LSP gets notified of the event only after the Dialog Manager has tried to run your User Proc, which will rely on globals that have become invalid since your program is switched out. Anyway, I nearly got my hard-disk trashed while trying to figure out the problem!

Note that the RMaker source file for the program doesn't include definitions for the KCAP resources used by the program. So after compiling this file with RMaker, you should use ResEdit to copy these resources from the Key Layout file in your system folder and paste them into your resource file. While you're at it, you could also install BNDL or SIZE resources to make the program look more fancy. It should run at ease under a 50K partition. There are many more possible improvements, but I'll leave them to your imagination.

Listing: KeyEdDemo.pas

```
PROGRAM KeyEdDemo;
{ copyright F.Samuel and MacTutor 1988 }
{ use only with system 4.x or later }
{$I-}
 { turn off automatic initialization }
{$L keyEdDemoRes}
 { load resource file }

  CONST
    DialogID = 128;
    EditCodeItem = 2;
    LoadBtn = 5;
    UserItem = 6;
    AsciiCharItem = 7;
    KeyCodeItem = 8;
    AboutAlrt = 129;
    LoadAlrt = 130;
    AppleID = 1;
    AboutItem = 1;
    FileID = 2;
    EditID = 3;
    CutItem = 3;
    CopyItem = 4;
    PasteItem = 5;
    ClearItem = 6;

{ Low-memory globals }
    KybdType = $21E;
    ScsiFlag = $B22;
    Key1Trans = $29E;
    BasicGlob = $2B6;

{ KCaps ID of various keyboards }
    MacPlusKbd = 11;
    MacClassicKbd = 3;
    EuroMacKbd = 259;
    ADBKbd = 1;
    ADBExtKbd = 2;
    ADBIsoKbd = 4;

  TYPE
    Prect = ^Rect; { for type-casting a pointer }
    PLong = ^LongInt;
    PWord = ^integer;
    PPoint = ^Point;

  VAR
    Finished, EditOn : Boolean;
    DragRect : Rect;
    MouseLocal : Point;
    KCapsHandle, KChrHandle : Handle;
    DemoDialog : DialogPtr;
    HiliteKeys : SET OF 0..127;
    EditKey, EditModifs : Integer;

{ utilities to acces properties of items in a dialog }

  PROCEDURE SetDItemText (TheDialog : DialogPtr;
                  TheItem : Integer;
                  TheText : Str255);
    VAR
      ItemType : integer; { should be a text item }
      ItemHandle : Handle;
      DispRect : Rect;
  BEGIN
    GetDItem(TheDialog, TheItem, ItemType, ItemHandle,
DispRect);
    SetIText(ItemHandle, TheText)
  END;

  FUNCTION GetDItemText (TheDialog : DialogPtr;
                  TheItem : Integer) : Str255;
    VAR
      ItemType : integer; { should be a text item }
      ItemHandle : Handle;
      DispRect : Rect;
      TheText : Str255;
  BEGIN
    GetDItem(TheDialog, TheItem, ItemType, ItemHandle,
DispRect);
    GetIText(ItemHandle, TheText);
    GetDItemText := TheText
  END;

  FUNCTION GetDItemRect (TheDialog : DialogPtr;
                  TheItem : Integer) : Rect;
    VAR
      ItemType : integer;
      ItemHandle : Handle;
      DispRect : Rect;
  BEGIN
    GetDItem(TheDialog, TheItem, ItemType, ItemHandle,
DispRect);
    GetDItemRect := DispRect
  END;

  PROCEDURE SetUserProc (TheDialog : DialogPtr;
                  TheItem : Integer;
                  TheProc : procPtr);
    VAR
      ItemType : integer; { should be an UserItem ! }
      ItemHandle : Handle;
      DispRect : Rect;
  BEGIN
    GetDItem(TheDialog, TheItem, ItemType, ItemHandle,
DispRect);
```

```
    SetDItem(TheDialog, TheItem, ItemType, Handle(theProc),
DispRect)
  END;

{ function NumToString , more convenient that way ... }

  FUNCTION FNumToString (TheNum : LongInt) : Str255;
    VAR
      TheString : Str255;
  BEGIN
    NumToString(TheNum, TheString);
    FNumToString := TheString
  END;

{ interface for external procedures and functions : }

  FUNCTION KeyTrans (transData : Ptr;
                     keycode : INTEGER;
                     VAR state : LONGINT) : LONGINT;
  INLINE
    $A9C3;

  PROCEDURE poke (address : longint;
                  value : integer);
  external; { puts low byte of value at address }

  FUNCTION peek (address : longint) : integer;
  external; { returns byte at address }

  FUNCTION Key12Trans (KeyCode, KeyModifs : Integer) :
Integer;
  external;

{ Pascal procedures and functions follow ... }

  FUNCTION GetAscii (KeyCode, Modifs : integer;
                     VAR Ascii : integer) : Boolean;
   { Returns true if it's a normal key , Ascii returns ascii
code even if it's a double strike}
    VAR
      State : LongInt;
  BEGIN
    State := 0;
    Ascii := LoWord(KeyTrans(KChrHandle^, KeyCode + Modifs,
State));
    IF Ascii = 0 THEN
      BEGIN
        GetAscii := false;
        Ascii := LoWord(KeyTrans(KChrHandle^, KeyCode +
Modifs, State))
      END
    ELSE
      GetAscii := true
  END;

  FUNCTION GetKbd : Integer;
  { returns KCAP ID of current keyboard }
    VAR
      TempID : integer;
      addr : Plong;
      SCSIMac : boolean;
  BEGIN
    TempId := peek(KybdType);
    SCSIMac := BitTst(Ptr(SCSIFlag), 5);
    IF (NOT SCSIMac) AND (TempID <> MacPlusKbd) THEN
      BEGIN
        tempId := MacClassicKbd;
        addr := Plong(Key1Trans);
        IF peek(addr^ + 10) <> 0 THEN { test itlc byte }
          TempId := EuroMacKbd
      END;
    GetKbd := TempID
  END;

PROCEDURE SetUpMenus;
  VAR
    ID : integer;
BEGIN
  FOR ID := AppleID TO EditID DO
    InsertMenu(GetMenu(ID), 0);
  AddResMenu(GetMHandle(AppleID), 'DRVR');
  DrawMenuBar
END;

PROCEDURE MainCaps (PROCEDURE treatIt (rgn : RgnHandle;
                                       Code : integer));
  VAR
    Hrgn : RgnHandle;
    addr : PWord;
    Paddr : PPoint;
    NumRgn, NumRect, NumKeys, i, j : integer;
    Keycode, dh, dv : Integer;
    TL, BR : point;
    KRect : rect;
BEGIN
  BEGIN
    SetPort(DemoDialog);
    GetMouse(MouseLocal);
    ClipRect(DemoDialog^.PortRect);
    Hlock(KcapsHandle);
    Addr := Pword(Ord4(KcapsHandle^) + 16);
    NumRgn := Addr^;
    IF NumRgn > 0 THEN
      FOR i := 1 TO NumRgn DO
        BEGIN
          Addr := Pword(Ord4(Addr) + 2);
          NumRect := addr^;
          Hrgn := NewRgn;
          OpenRgn;
          SetPt(TL, 0, 0);
          Addr := Pword(Ord4(Addr) + 2);
          FOR j := 0 TO NumRect DO
            BEGIN
              PAddr := PPoint(addr);
              BR := Paddr^;
              Pt2Rect(TL, BR, Krect);
              FrameRect(Krect);
              TL := BR;
              Addr := Pword(Ord4(Addr) + 4);
            END;
          CloseRgn(Hrgn);
          NumKeys := addr^;
          FOR j := 0 TO NumKeys DO
            BEGIN
              Addr := Pword(Ord4(Addr) + 2);
              KeyCode := addr^;
              Addr := Pword(Ord4(Addr) + 2);
              dv := addr^;
              Addr := Pword(Ord4(Addr) + 2);
              dh := addr^;
              OffsetRgn(Hrgn, dh, dv);
              TreatIt(Hrgn, Keycode MOD 128);
            END;
          DisposeRgn(Hrgn);
        END;
      Hunlock(KcapsHandle)
    END
  END;

  PROCEDURE InvertKey (Rgn : RgnHandle);
    VAR
      InnerRgn : RgnHandle;
  BEGIN
    InnerRgn := NewRgn;
    CopyRgn(Rgn, InnerRgn);
    InsetRgn(InnerRgn, 2, 2);
    InvertRgn(InnerRgn);
    DisposeRgn(InnerRgn)
```


© The Definitive MacTutor, Vol. 4

```
  END;

  PROCEDURE DrawKey (rgn : RgnHandle;
                     Code : integer);
    VAR
      DrawRgn : RgnHandle;
      AsciiCode : Integer;
      NormalKey : boolean;
  BEGIN
    FrameRgn(Rgn);
    DrawRgn := NewRgn;
    CopyRgn(Rgn, DrawRgn);
    InsetRgn(DrawRgn, 1, 1);
    SetClip(DrawRgn);
    EraseRgn(DrawRgn);
    NormalKey := GetAscii(Code, EditModifs, AsciiCode);
    WITH DrawRgn^^.rgnBBox DO
      MoveTo(left + 1, bottom - 2);
    DrawChar(Chr(AsciiCode));
    DisposeRgn(DrawRgn);
    IF Code IN HiliteKeys THEN
      InvertKey(Rgn);
    ClipRect(DemoDialog^.PortRect)
  END;

  PROCEDURE UserDraw (TheWindow : WindowPtr;
                      ItemNum : Integer);
    VAR
      FillPat : Pattern;
      TheRect : Rect;
  BEGIN
    TheRect := GetDItemRect(DemoDialog, ItemNum);
    GetIndPattern(FillPat, 0, 10);
    FillRect(TheRect, FillPat);
    FrameRect(TheRect);
    MainCaps(DrawKey)
  END;

  PROCEDURE InitThings;
    VAR
      i, Error : Integer;
  BEGIN { Get KybdID and KCaps ; SetUserProc;Show dialog }
    FlushEvents(EveryEvent, 0);
    InitGraf(@ThePort);
    InitFonts;
    InitWindows;
    InitMenus;
    TEInit;
    InitDialogs(NIL);
    InitCursor;
    SetEventMask(EveryEvent - KeyUpMask);
    MaxApplZone;
    FOR i := 1 TO 5 DO
      MoreMasters;
    WITH ScreenBits.Bounds DO
      SetRect(DragRect, left + 4, top + 24, right - 4, bottom
- 4);
    SetUpMenus;
    KCapsHandle := GetResource('KCAP', GetKbd);
    DetachResource(KCapsHandle); { don't let a DA release it
on your back ! }
    KChrHandle := GetResource('KCHR', 0);
    Error := HandToHand(KChrHandle); { make a copy , so we can
modify it }
    MoveHHi(KChrHandle); { keep heap unfragmented }
    HLock(KChrHandle);
    HiliteKeys := []; { no keys to hilite until the user
selects one }
    EditModifs := 0;
    EditOn := False; { wait for the user to select a key to
edit }
    DemoDialog := GetNewDialog(DialogID, NIL, WindowPtr(-1));
    SetUserProc(DemoDialog, UserItem, @UserDraw);
    ShowWindow(DemoDialog);
    Finished := False
  END;

  PROCEDURE DoMenu (Code : longint);
    VAR
      MenuNum, ItemNum, Temp : integer;
      DeskAccName : str255;
  BEGIN
    IF code <> 0 THEN
      BEGIN
        MenuNum := HiWord(Code);
        ItemNum := LoWord(Code);
        CASE MenuNum OF
          AppleID :
            IF ItemNum = AboutItem THEN
              Temp := Alert(AboutAlrt, NIL)
            ELSE
              BEGIN
                GetItem(GetMHandle(AppleID), ItemNum,
DeskAccName);
                Temp := OpenDeskAcc(DeskAccName);
              END;
          FileID :
            Finished := True;
          EditID :
            IF NOT SystemEdit(ItemNum - 1) THEN
              IF (FrontWindow = DemoDialog) AND EditOn THEN
                CASE ItemNum OF
                  CutItem :
                    DlgCut(DemoDialog);
                  CopyItem :
                    DlgCopy(DemoDialog);
                  PasteItem :
                    DlgPaste(DemoDialog);
                  ClearItem :
                    DlgDelete(DemoDialog);
                  OTHERWISE
                END;
          OTHERWISE
        END;
        HiliteMenu(0)
      END
  END;

  PROCEDURE StartEdit (KeyCode, AsciiCode : Integer);
  { enable editing  the ascii code of the selected key }
  BEGIN
    EditKey := KeyCode;
    HiliteKeys := HiliteKeys + [EditKey];
    SetDItemText(DemoDialog, EditCodeItem,
FNumToString(AsciiCode));
    SelIText(DemoDialog, EditCodeItem, 0, MaxInt);
    SetDItemText(DemoDialog, KeyCodeItem,
FNumToString(EditKey));
    SetDItemText(DemoDialog, AsciiCharItem, Chr(AsciiCode));
    EditOn := true
  END;

  PROCEDURE DrawEditKRgn (Rgn : RgnHandle;
                          TheCode : Integer);
  { update key contents in case it changed }
  BEGIN
    IF TheCode = EditKey THEN
      DrawKey(Rgn, EditKey)
  END;

  PROCEDURE ValidateEdit;
  { validate user editing of the selected key }
    VAR
      BlockNumber : Integer;
      MapAddress, NewAsciiCode : LongInt;
  BEGIN
    IF EditOn THEN
      BEGIN
```

```
        StringToNum(GetDItemText(DemoDialog, EditCodeItem),
NewAsciiCode);
        MapAddress := Ord4(KChrHandle^);
        BlockNumber := Peek(MapAddress + BitShift(EditModifs,
-8) + 2);
        Poke(MapAddress + 260 + BlockNumber * 128 + EditKey,
NewAsciiCode);
        EditOn := False;
        SetDItemText(DemoDialog, EditCodeItem, '');
        SetDItemText(DemoDialog, KeyCodeItem, '');
        SetDItemText(DemoDialog, AsciiCharItem, '');
        HiliteKeys := HiliteKeys - [EditKey];
        MainCaps(DrawEditKRgn);
      END
  END;

  PROCEDURE CodeXORModifs (ModifCode : Integer;
                   VAR Modifs : Integer);
  { XOR new modifier with the already selected ones }
    VAR
      TempModifs : Integer;
  BEGIN
    TempModifs := 0;
    BitSet(@TempModifs, $3E - ModifCode);
    Modifs := BitXor(Modifs, TempModifs)
  END;

  FUNCTION EditPerm (KeyCode, Modifs : Integer;
                   VAR AsciiCode : Integer) : Boolean;
{ can that key  be edited ? }
  BEGIN
    IF GetAscii(KeyCode, Modifs, AsciiCode) THEN
      EditPerm := true
    ELSE { check that it's not a double strike nor a modifier
}
      EditPerm := (AsciiCode = 0) AND NOT (KeyCode IN
[$3C..$3E])
  END;

  PROCEDURE ActiveClick (rgn : RgnHandle;
                   Code : integer);
    VAR
      AsciiCode : Integer;
  BEGIN
    IF PtInRgn(MouseLocal, Rgn) THEN
      IF (Code <> EditKey) OR NOT EditOn THEN
        BEGIN
          ValidateEdit;
          IF Code IN [$37..$3B] THEN { modifier key was
clicked }
            BEGIN
              CodeXORModifs(Code, EditModifs);
              IF Code IN HiliteKeys THEN
                HiliteKeys := HiliteKeys - [Code]
              ELSE
                HiliteKeys := HiliteKeys + [Code];
              IF EditPerm(EditKey, EditModifs, AsciiCode)
THEN
                StartEdit(EditKey, AsciiCode);
              MainCaps(DrawKey)  { redraw the keyboard to
update hiliting }
            END
          ELSE IF EditPerm(Code, EditModifs, AsciiCode) THEN
            BEGIN
              InvertKey(Rgn); { hilite it , and let user edit
it }
              StartEdit(Code, AsciiCode)
            END
        END
  END;

  FUNCTION CheckSysTrans (TransData : Ptr) : Boolean;
   { is TransData really a pointer to the system mapping table
?? }
    VAR
      KeyCode, AsciiCode : Integer;
  BEGIN
    CheckSysTrans := true;
    FOR KeyCode := 0 TO 16 DO
      BEGIN
        AsciiCode := Key12Trans(KeyCode, 0); { let system
compute it }
        IF Peek(Ord4(TransData) + 260 + KeyCode) <> AsciiCode
THEN
          CheckSysTrans := False; { compare with our table }
        IF AsciiCode = 0 THEN
          BEGIN
            AsciiCode := Key12Trans(KeyCode, 0);
            FlushEvents(EveryEvent, 0) { double strike may
have posted an event , flush it }
          END
      END
  END;

  FUNCTION GetSysTrans (VAR TheTrans : Ptr) : boolean;
    VAR
      BGlob, Addr : PLong;
  BEGIN
    Bglob := PLong(BasicGlob);
    Addr := PLong(BGlob^ + 14);
  { not documented , so better check if we can rely on it ! }
    TheTrans := Ptr(Addr^);
    GetSysTrans := CheckSysTrans(TheTrans)
  END;

  PROCEDURE DoLoad;
    VAR
      TheSize : LongInt;
      RamTransPtr : Ptr;
      SysKchr : Handle;
      AppResFile : integer;
  BEGIN
    IF Alert(LoadAlrt, NIL) = Ok THEN
      BEGIN
        TheSize := GetHandleSize(KChrHandle);
        IF GetSysTrans(RamTransPtr) THEN { see above ... }
          BlockMove(KChrHandle^, RamTransPtr, TheSize);
        AppResFile := CurResFile;
        UseResFile(0);
        SysKchr := GetResource('KCHR', 0);
        BlockMove(KChrHandle^, SysKChr^, TheSize);
        ChangedResource(SysKChr);
        UpdateResFile(0);
        UseResFile(AppResFile)
      END
  END;

  PROCEDURE DoDialog (TheEvent : EventRecord);
    CONST
      Enter = $03;
      Return = $0D;
    VAR
      TheDialog : DialogPtr;
      ItemHit, CharCode : Integer;
      PassIt : Boolean;
  BEGIN
    WITH TheEvent DO
      IF What = KeyDown THEN
        BEGIN { filter key down events }
          CharCode := BitAnd(Message, CharCodeMask);
          PassIt := False;
          IF BitAnd(Modifiers, CmdKey) <> 0 THEN
            DoMenu(MenuKey(Chr(CharCode)))
          ELSE IF EditOn THEN
            IF CharCode IN [Return, Enter] THEN
              ValidateEdit
            ELSE
              PassIt := True
```

```
        END
      ELSE
        PassIt := True;
    IF PassIt THEN
      IF DialogSelect(TheEvent, TheDialog, ItemHit) THEN
        CASE ItemHit OF
          UserItem :
            MainCaps(ActiveClick);
          LoadBtn :
            DoLoad;
          OTHERWISE
        END
  END;

  PROCEDURE MainLoop;
    VAR
      GotEvent : Boolean;
      TheEvent : EventRecord;
      TheWindow : WindowPtr;
  BEGIN
    SystemTask;
    GotEvent := GetNextEvent(EveryEvent, TheEvent);
    IF IsDialogEvent(TheEvent) THEN
      DoDialog(TheEvent)
    ELSE IF GotEvent THEN
      WITH TheEvent DO
        CASE What OF
          MouseDown :
            CASE FindWindow(Where, TheWindow) OF
              inMenuBar :
                DoMenu(MenuSelect(Where));
              inSysWindow :
                SystemClick(TheEvent, TheWindow);
              inContent :
                IF TheWindow <> FrontWindow THEN
                  SelectWindow(TheWindow);
              inDrag :
                DragWindow(TheWindow, Where, DragRect);
              inGoaway :
                IF TheWindow <> FrontWindow THEN
                  SelectWindow(TheWindow)
                ELSE IF TrackGoAway(TheWindow, Where) THEN
                  Finished := True;
              OTHERWISE
            END;
          KeyDown :
            IF BitAnd(Modifiers, CmdKey) <> 0 THEN
              DoMenu(MenuKey(Chr(BitAnd(Message, CharCode-
Mask))));
          UpdateEvt :
            BEGIN { just in case ... }
              TheWindow := WindowPtr(Message);
              BeginUpdate(TheWindow);
              EndUpdate(TheWindow)
            END;
          OTHERWISE
        END
  END;

BEGIN
  InitThings;
  REPEAT
    MainLoop
  UNTIL Finished
END.
```

Listing: KeyEdDemo.asm

```
;MDS source code for KeyEDDemo External procedures
;copyright F.Samuel and Mac Tutor 1988
; After compiling , turn the .rel file into a LSP
; library using rel.Converter , then include the lib
; into your LSP project

;MDS Glues for Pascal Peek and Poke
;procedure poke(address:ptr;value:byte)
;function peek(address:ptr): byte;

XDEF peek
XDEF poke

value       EQU  8
addr1       EQU  10

poke
  LINK      A6,#0
  MOVEA.L   addr1(A6),A0
  MOVE      value(A6),D0
  MOVE.B    D0,(A0)
  UNLK      A6
  MOVEA.L   (SP)+,A0
  ADDQ.L    #6,SP
  JMP       (A0)

addr2       EQU  8
result      EQU  12

peek
  LINK A6,#0
  MOVEA.L  addr2(A6),A0
  MOVEQ  #0,D0
  MOVE.B (A0),D0
  MOVE D0,result(A6)
  UNLK A6
  MOVEA.L  (SP)+,A0
  ADDQ.L #4,SP
  JMP  (A0)

; Function Key12Trans(KeyCode,KeyModifs:Integer) : Integer;
; KeyModifs are as they appear in the third word of KeyMap
; Function returns AsciiCode

XDEF          Key12Trans

Key1Trans   EQU  $29E
Key2Trans   EQU  $2A2

KeyModifs   EQU  8
KeyCode     EQU  10
FunRslt     EQU  12

Key12Trans

  LINK      A6,#0
  MOVEM.L   A0/D0-D3,-(SP)
  MOVE      KeyCode(A6),D2
  MOVE      KeyModifs(A6),D1
  MOVEQ     #0,D3
  CMPI      #64,D2
  BHS.S     @1
  MOVEA.L   Key1Trans,A0
  BRA.S     @2
@1MOVEA.L   Key2Trans,A0
@2JSR       (A0)
  MOVE      D0,FunRslt(A6)
  MOVEM.L   (SP)+,A0/D0-D3
  UNLK      A6
  MOVEA.L   (SP)+,A0
  ADDQ.L    #4,SP
  JMP       (A0)
```

Listing: KeyEdDemoRes.R

```
* KeyEdDemo resource definitions for RMaker *
* Copyright 1988 - F.Samuel and Mac Tutor   *
* After compiling , don't forget to add the *
* KCAP resources from Key Layout in your    *
* System Folder , using ResEdit     . Then      *
```

```
* include the file into your LSP project    *
* using Run Options from the Project Menu   *

KeyEdDemoRes

Type MENU

     ,1
\14
About KeyEdDemo
(-

     ,2
File
Quit/Q

     ,3
Edit
Undo/Z
(-
Cut/X
Copy/C
Paste /V
Clear

Type DLOG

     ,128
Key Ed Demo
76 14 306 504
Invisible GoAway
4 ;definition ID
0 ;ref con
128 ;DITL ID

Type ALRT

     ,129
42 102 94 410
129 ;DITL ID
CCCC ;Alert stages

     ,130
42 136 142 376
130
5555

Type DITL

     ,128
8
*   1
StatText Disabled
5 3 25 89
Ascii Code :

*   2
EditText Disabled
5 96 25 133

*   3
StatText Disabled
5 279 25 357
Key Code :

*   4
StatText Disabled
5 144 25 221
Ascii Char :

*   5
BtnItem Enabled
5 418 25 478
Load

*   6
UserItem Enabled
32 5 222 482

*   7
StatText Disabled
5 223 25 257

*   8
StatText Disabled
5 360 25 393

     ,129
1
*   1
StatText Enabled
5 3 48 305
Key Ed Demo\0DCopyright Fabien Samuel and Mac Tutor 1988

     ,130
3
*   1
BtnItem Enabled
75 6 95 66
OK

*   2
BtnItem Enabled
75 170 95 230
Cancel

*   3
StatText Disabled
6 5 65 232
Are you sure you want to load this mapping into the System
File and into RAM ?
```

© The Definitive MacTutor, Vol. 4

Basic School

Basic School
Color & DragGrayRgn Explored

Dave Kelly
MacTutor Editorial Board
MacTutor Vol. 4 No. 1

MAC II takes on BASIC

Yes, I know that it has been about a year since the Macintosh II was released and not a word from me on how the Basic works on the Mac II. The Mac II really is a wonderful machine, although there are still a few incompatibilities and inconveniences. But all in all I still give the Mac II an A+.

The curiosity lies in figuring out what works (or doesn't work) on the Mac II. The Mac II has features that did not even exist when most of the versions of Basic were written. Some things like color and screen size aren't supported properly, or at all, even though the software was capable before the Mac II came.

It's fast. For example, just take a look at what it did for the Byte Magazine January 1983 Prime number benchmark program, known as the Sieve. ZBasic™ comes up with a 2 second time, MS Basic Binary interpreter with 129 second time (about the same as True Basic on the Mac Plus), and 15 seconds for the MS Basic compiler. Of course, everything is faster so it is all relative. Since we already know that the Mac II is faster and have a good idea about which versions of Basic are faster, there is no need to go into that here. For more information comparing the various versions of Basic, see MacTutor, Aug 1986, Feb. 1987 and Mar. 1987.

Screen Size: With the advent of the larger screen (also available for Mac Plus and SE) we see the need to be fully compatible with all size monitors. MS Basic opens its default windows to 512 X 342, the size of the SE and MacPlus screens. That works out alright since it still isn't known what kind of Mac might be running the program. Double-clicking the title bar causes MS Basic windows to zoom to full screen size. The LIST windows won't expand horizontally beyond the 512 limit, but will expand vertically to the bottom of the screen. The most annoying thing is that none of the windows can be moved below a certain point on the screen which is very close to the place where the bottom of the 9" screen would be (342). This means that you can never move the command window out of the way; you have to close it just like on a Mac Plus.

True Basic v. 1.1, which I haven't looked at for awhile, opens the default window to full screen size. Of course, True Basic still has all the HFS problems and other limitations which I have discussed in past issues of MacTutor (Best of MacTutor Vol 2, pg 379 or August 1986). (Version 1.1 is the latest that I have).

ZBasic Ver. 4.1 windows also open to full screen size. There doesn't appear to be any compatibility problems here at all. As I write this, the ZBasic promised editor has still not been completed.

Color support: This is only a big deal to those of you with a Mac II with a color monitor (I wouldn't have one any other way!!). As you know, the original Quickdraw supports 8 colors already. I feel that it shows short-sightedness to not at least support that much. But both MS Basic and True Basic have NO color support even though both languages include means to change color from black to white. Other versions (on other computers whose names we won't say) of True Basic support a small set of colors. MS Basic uses color as a parameter in the syntax for the CIRCLE statement, yet you can't get any colors except black and white.

ZBasic, however, does support the 8 colors in Quickdraw. That's only 8 colors for now, but I would expect that they will make improvements to this in some later version (who knows when?). So meanwhile, if you want color you will have to settle with the 8 basic colors. There are a few differences between ZBasic color statements and Quickdraw color statements. In order to standardize ZBasic, Zedcor has numbered the colors from 0 to 7. Quickdraw, however, has its own designation for each color as the following chart shows:

COLOR TABLE

COLOR	ZBasic	QuickDraw
White	0	25
Yellow	1	130
Green	2	190
Cyan	3	273
Blue	4	305
Magenta	5	137
Red	6	205
Black	7	10

The color codes in ZBasic and QuickDraw CANNOT be substituted for each other. You will have to either use Quickdraw graphics or ZBasic graphics or be careful not to mix the code with the wrong call. Sure, setting the color is easy, but be sure to use the right code with the right statement.

This month's program demonstrates the colors available in ZBasic and gives you an example of its use. All you have to do

© The Definitive MacTutor, Vol. 4

is set the color before issuing a graphics drawing statement and either Quickdraw or ZBasic (which probably uses Quickdraw anyway) will use the color you selected.

What's missing? Color QuickDraw™ that's what. Fortunately, Zedcor has been very responsive to updating ZBasic. I'm sure that it isn't easy for them to keep updating everyone so often. Let's hope that the next version of ZBasic will include the long awaited editor, Color QuickDraw and other major improvements. Read on!

[The slow response by the Basic compiler makers to upgrade their products to meet the requirements of Mutifinder, System 4.2 and the Macintosh II, has been really disappointing! There is still a pressing need for a reliable, engineering oriented Basic that provides full access to the Macintosh capabilities and can function in a workstation environment (ie, it can do math correctly and quickly). We encourage both the present Basic compiler makers to address this issue, as well as those we haven't seen in this marketplace, like Borland, to enter it, and hopefully unite a fragmented and frustrated market. -Ed]

```
'ColorBasic
'MacTutor, 1988
'By Dave Kelly

WINDOW OFF
COORDINATE WINDOW
'A color chart for future reference
DIM rect%(3)
MENU 1,0,1,"File"
MENU 1,1,1,"Quit"
EDIT MENU 2
MENU 3,0,1,"Colors"
MENU 3,1,1,"ZBasic™ Commands"
MENU 3,2,1,"ToolBox Commands"
MENU 3,3,1,"Erase window"
theEnd=0:REM theEnd will never come.

'Make QuickDraw Color Assignments
blackColor&=33
whiteColor&=30
redColor&=205
greenColor&=341
blueColor&=409
cyanColor&=273
magentaColor&=137
yellowColor&=69

'Make ZBasic Color Assignments
white=0
yellow=1
green=2
cyan=3
blue=4
magenta=5
red=6
black=7
WINDOW 1,"Color Demo Window",(4,38)-(450,200),4
GOSUB "ZBasic™ Colors": ' Print something to start
ON MENU GOSUB "MenuEvent"
MENU ON
theEnd=0:REM theEnd will never come.
DO
    UNTIL theEnd

MENU OFF

"MenuEvent"
Menunumber%=MENU(0)
Menuitem%=MENU(1)
MENU
IF Menunumber%=1 AND Menuitem%=1 THEN "Quit"
IF Menunumber%=3 AND Menuitem%=1 THEN
            GOSUB "ZBasic™ Colors"
IF Menunumber%=3 AND Menuitem%=2 THEN
            GOSUB "QuickDraw™ Colors"
IF Menunumber%=3 AND Menuitem%=3 THEN
CLS:PRINT@(1,3)"The Colors are now cleared!"
RETURN

"Quit"
END

"ZBasic™ Colors"
CLS
TEXT 2,14,0,0
PRINT%(170,20) "ZBasic™ Colors"
COLOR=white
BOX FILL 5,50 TO 55,100
COLOR=yellow
BOX FILL 60,50 TO 110,100
COLOR=green
BOX FILL 115,50 TO 165,100
COLOR=cyan
BOX FILL 170,50 TO 220,100
COLOR=blue
BOX FILL 225,50 TO 275,100
COLOR=magenta
BOX FILL 280,50 TO 330,100
COLOR=red
BOX FILL 335,50 TO 385,100
COLOR=black
BOX FILL 390,50 TO 440,100
BOX 5,50 TO 55,100
BOX 60,50 TO 110,100
BOX 115,50 TO 165,100
BOX 170,50 TO 220,100
BOX 225,50 TO 275,100
BOX 280,50 TO 330,100
BOX 335,50 TO 385,100
```

```
BOX 390,50 TO 440,100
TEXT 2,10
PRINT%(25,48) white
PRINT%(75,48) yellow
PRINT%(130,48) green
PRINT%(190,48) cyan
PRINT%(240,48) blue
PRINT%(300,48) magenta
PRINT%(355,48) red
PRINT%(410,48) black
PRINT%(25,112) whiteColor&
PRINT%(75,112) yellowColor&
PRINT%(130,112) greenColor&
PRINT%(190,112) cyanColor&
PRINT%(240,112) blueColor&
PRINT%(300,112) magentaColor&
PRINT%(350,112) redColor&
PRINT%(410,112) blackColor&
TEXT 2,14
PRINT%(150,140) "QuickDraw™ Colors"
RETURN

"QuickDraw™ Colors"
CLS
CALL MOVETO (170,20)
CALL TEXTFONT(2)
CALL TEXTSIZE(14)
CALL DRAWSTRING ("ZBasic™ Colors")
CALL SETRECT(rect%(0),5,50,55,100)
CALL FORECOLOR(whiteColor&)
CALL PAINTRECT(rect%(0))
CALL SETRECT(rect%(0),60,50,110,100)
CALL FORECOLOR(yellowColor&)
CALL PAINTRECT(rect%(0))
CALL SETRECT(rect%(0),115,50,165,100)
CALL FORECOLOR(greenColor&)
CALL PAINTRECT(rect%(0))
CALL SETRECT(rect%(0),170,50,220,100)
CALL FORECOLOR(cyanColor&)
CALL PAINTRECT(rect%(0))
CALL SETRECT(rect%(0),225,50,275,100)
CALL FORECOLOR(blueColor&)
CALL PAINTRECT(rect%(0))
CALL SETRECT(rect%(0),280,50,330,100)
CALL FORECOLOR(magentaColor&)
CALL PAINTRECT(rect%(0))
CALL SETRECT(rect%(0),335,50,385,100)
CALL FORECOLOR(redColor&)
CALL PAINTRECT(rect%(0))
CALL SETRECT(rect%(0),390,50,440,100)
CALL FORECOLOR(blackColor&)
CALL PAINTRECT(rect%(0))
CALL SETRECT(rect%(0),5,50,55,100)
CALL FRAMERECT(rect%(0))
CALL SETRECT(rect%(0),60,50,110,100)
CALL FRAMERECT(rect%(0))
CALL SETRECT(rect%(0),115,50,165,100)
CALL FRAMERECT(rect%(0))
CALL SETRECT(rect%(0),170,50,220,100)
CALL FRAMERECT(rect%(0))
CALL SETRECT(rect%(0),225,50,275,100)
CALL FRAMERECT(rect%(0))
CALL SETRECT(rect%(0),280,50,330,100)
CALL FRAMERECT(rect%(0))
CALL SETRECT(rect%(0),335,50,385,100)
CALL FRAMERECT(rect%(0))
CALL SETRECT(rect%(0),390,50,440,100)
CALL FRAMERECT(rect%(0))
CALL TEXTSIZE (10)
CALL MOVETO (25,48)
PRINT white
CALL MOVETO (75,48)
PRINT yellow
CALL MOVETO (130,48)
PRINT green
CALL MOVETO (190,48)
PRINT cyan
CALL MOVETO (240,48)
PRINT blue
CALL MOVETO (300,48)
PRINT magenta
CALL MOVETO (355,48)
PRINT red
CALL MOVETO (410,48)
PRINT black
CALL MOVETO (25,112)
PRINT whiteColor&
CALL MOVETO (75,112)
PRINT yellowColor&
CALL MOVETO (130,112)
PRINT greenColor&
CALL MOVETO (190,112)
PRINT cyanColor&
CALL MOVETO (240,112)
PRINT blueColor&
CALL MOVETO (300,112)
PRINT magentaColor&
CALL MOVETO (350,112)
PRINT redColor&
CALL MOVETO (410,112)
PRINT blackColor&
CALL TEXTSIZE(14)
CALL MOVETO (150,140)
CALL DRAWSTRING ("QuickDraw™ Colors")
RETURN
```

Basic Answers to Basic Questions

Concerning the comment in the Nov. 1987 issue of MacTutor where Steve Millman suggested a method of applying an

event loop for reading disk events: The main problem here is that a disk event might be inserted while not in the GetNextEvent loop. The disk insert would be missed completely. I have been assured by Zedcor that in a future release of ZBasic they will add the Disk insert event to the DIALOG functions. This is easy for them to do since GETNEXTEVENT returns this to them anyway. For now, just do what Steve Millman suggests or don't bother checking for disks at all until the new improvements come.

Q. I want to use in my program a **DragGrayRect** statement (the same as you see in MS Basic), but I want to use it in ZBasic. Is this possible?

A. **DragGrayRect** is only available as an MS Basic library call (previously known as a Clear Lake Research Library call). The alternative is to use **DragGrayRgn** where the region is in the shape of a rectangle. **DragGrayRgn** is a ROM call where as **DragGrayRect** is a library subroutine derived from ROM calls (probably uses **DragGrayRgn**). Another alternative that would require more work would be to write your own **DragGrayRect** function to use in any of your programs. The following program includes an example of this function. The function itself can be saved to the disk with SAVE+ and retrieved later for use in other programs with APPEND. Hope that this helps.

```
'Drag Example
'By Dave Kelly
'©MacTutor, 1988

WINDOW OFF
COORDINATE WINDOW
DEF MOUSE=-1
DIM Rect%(3),pt%(1),pin%(3),bnd%(3),dir%,dis&
pt%(0)=mousey%:pt%(1)=mousex%
'Create DragGrayRect function
LONG FN DragGrayRect&(RL%,RT%,RR%,RB%,
mousey,mousex, PL%,PT%,PR%,PB%,BL%,BT%,BR%,
BB%,dir%)
'    Rect%  =  the rectangle to be moved
'    pt%    =  the point where the mouse was pushed.
'    pin%   =  the rectangular limits in which outline can be
'              dragged.
'    bnd%   =  the rectangular boundary for a drag.
'    dir%=  the direction to which the drag is constrained
'              0 = no constraint, 1 = horizontal, 2 = vertical
'    dis&=  A point array that returns drag displacement.
    CALL SETRECT(Rect%(0),RL,RT,RR,RB)
    CALL SETRECT(pin%(0),PL,PT,PR,PB)
    CALL SETRECT(bnd%(0),BL,BT,BR,BB)
    RgnHand&=FN NEWRGN
    CALL OPENRGN
    CALL FRAMERECT(Rect%(0))
    CALL CLOSERGN(RgnHand&)
    pt%(0)=mousey%:pt%(1)=mousex%:Proc&=0:dis&=0
    dis&=FN DRAGGRAYRGN(RgnHand&, pt%(0),
pin%(0), bnd%(0), dir%,Proc&)
    CALL DISPOSERGN(RgnHand&)
END FN=dis&
'Find out monitor size
CALL GETWMGRPORT(WMgrPort&)
PortTop=PEEK WORD(WMgrPort&+8)
PortLeft=PEEK WORD(WMgrPort&+10)
PortBottom=PEEK WORD(WMgrPort&+12)
PortRight=PEEK WORD(WMgrPort&+14)
WINDOW#1,"DragExample",(PortLeft+4,PortTop+42)-
          (PortRight-6, PortBottom-6),5
MENU 1,0,1,"File"
MENU 1,1,1,"Quit"
top=100:left=100
GOSUB "DrawBox"
MOUSE ON:MENU ON:DIALOG ON
ON MOUSE GOSUB "MouseEvent"
ON MENU GOSUB "MenuEvent"
ON DIALOG GOSUB "DialogEvent"
"Loop"
DO
UNTIL Done
MOUSE OFF:MENU OFF:DIALOG OFF

"DialogEvent"
D=DIALOG(0)
IF D=4 THEN END
RETURN

"MenuEvent"
Menunumber=MENU(0)
Menuitem=MENU(1)
IF Menunumber=1 AND Menuitem=1 THEN END
MENU
RETURN

"MouseEvent"
X=MOUSE(0)
mousey=MOUSE(2):mousex=MOUSE(1)
dir%=0
dis&= FN DragGrayRect&(left,top, left+100, top+100, mousey,
    mousex, PortLeft, PortTop, PortRight, PortBottom,
    PortLeft, PortTop, PortRight,PortBottom,dir%)
dy=FN HIWORD(dis&)
dx=FN LOWORD(dis&)
LONG IF dx<>0 AND dy<>0
    CALL ERASERECT(Rect%(0))
    top=top+dy:left=left+dx
    GOSUB "DrawBox"
END IF
RETURN

"DrawBox"
CALL SETRECT(Rect%(0),left,top,left+100,top+100)
CALL FORECOLOR(205): ' color = red
CALL PAINTRECT(Rect%(0))
RETURN
```

Basic School
Updating Windows in Basic

Dave Kelly
MacTutor Editorial Board
MacTutor Vol. 4 No. 3

Dave Kelly is an electrical engineer for General Dynamics in Pomona, where he is involved in using work stations for hardware design. He is a founding editorial board member and currently an active board member for MacTutor.

A fresh window lets you see things more clearly. *(I like mine in color, but black & white mode on the Macintosh II sure is faster).* I received the following request from one of our readers which I'm sure others of you would also like to know about:

Q. I am working on an application in ZBasic 4.00 that I hope to release commercially. Previously I never had to worry about handling update events in a "correct" manner (in the past I have always reprinted everything on the screen when I've encountered an update event). ZBasic does provide a WINDOW PICTURE statement that I have not gotten to work to my satisfaction. I have also tried GET and PUT without much success. I'm obviously doing something wrong. ...could you show various ways to handle update events that involve DAs and multiple windows? My application has Edit Fields and displayed text (i.e. displaying results obtained from values entered in the edit fields).

A. Yes, there can be some tricky problems associated with refreshing windows which use edit fields. There is a standard way that refreshing should work. In Pascal the application should respond by calling BeginUpdate, drawing the window, then call EndUpdate. The Window Manager section of Inside Macintosh explains how to do this. That's fine when using the Macintosh standard GetNextEvent loop, but in ZBasic the approach must be modified.

The object is to restore any window which gets covered up by another window. If window 1 is dragged over window 2, then if window 2 is selected it should automatically be redrawn. Or if only part of window 2 is covered up and window 1 is removed then window 2 should be refreshed. This means that you as a programmer are responsible to determine all of the possible ways the window could be erased and be sure to set up event handling to cover all the possibilities.

There are 3 options we have for screen refresh with ZBasic.

1. Redraw the picture in response to a DIALOG event.
2. Use the WINDOW PICTURE statement.
3. Use GET and PUT in response to a DIALOG event.

By the end of this discussion you will know when to use each method in different circumstances. Directly redrawing the window is probably used most often by those that don't know

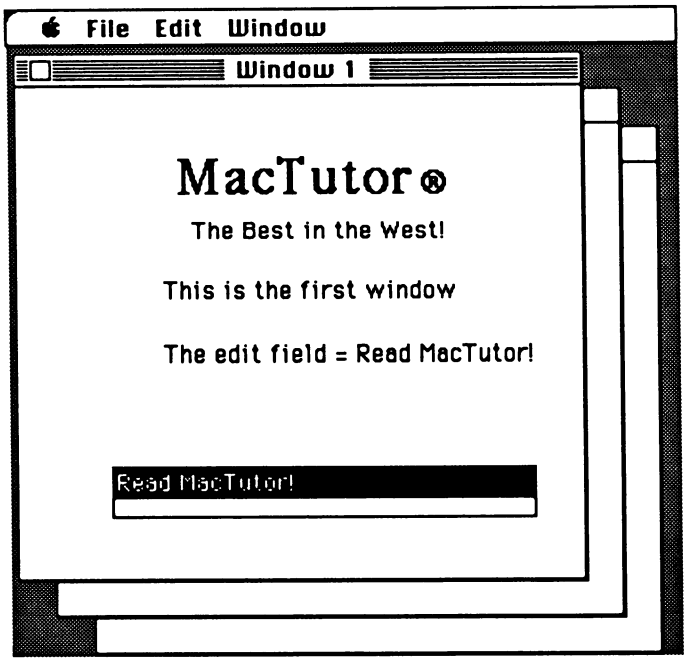


Fig. 1 Our Multiple Window Example

about the other two methods (or don't know how they work). Redrawing is what I think of as the "brute force" method. It is usually the method used for "quick and dirty" screen refresh. The advantage is that you have control to force the update to occur whenever you feel like it. The disadvantage is that it is slower and you may find that the screen may get updated sometimes when it doesn't need it (in order to catch every possible event).

To refresh by redrawing in the "brute force" mode a DIALOG(0)=5 event should effectively determine when a window needs to be refreshed. The window that needs to be refreshed is returned in the DIALOG(5) function. A quick call to the screen drawing routine when DIALOG event occurs is all that is needed to make this work.

The 2nd method of refreshing the screen is somewhat unique to ZBasic. Using the WINDOW PICTURE statement is the most automatic way to update the screen. A good example of using WINDOW PICTURE is given in the ZBasic manual on page E-158. The example at the end of this column also demonstrates its use. There are a few problems which you should be aware of. When using WINDOW PICTURE, all DIALOG events for the window involved are <u>not</u> passed to the DIALOG function, but instead are automatically handled by the WINDOW PICTURE statement. This is both good and bad. It is good because now any portion of the screen can be refreshed automatically without really trying. The problem comes when you use an EDIT FIELD

© The Definitive MacTutor, Vol. 4

in the window which you are using WINDOW PICTURE. When WINDOW PICTURE is active, the EDIT FIELD is not updated properly. Ordinarily the EDIT FIELD is automatically updated, but not with WINDOW PICTURE. This is one of those cases where the window should be refreshed with the "brute force" method. The process can be improved by setting up the screen information to be updated in a PICTURE so that it can be redrawn quickly. The demo shows how this is done. The "Set Window 1 Update" routine is called whenever an update event occurs. Also the EDIT FIELD value is re-read and the text displayed is obtained from the current contents of the EDIT FIELD. It is really too bad that in ZBasic the WINDOW PICTURE statement doesn't update controls. *[My guess is that WINDOW PICTURE is nothing more than the automatic window updating procedure of placing a quickdraw picture handle in the window record's pic field, in which case, no controls or edit fields would be updated. This method was only intended for fairly static window contents, and never meant to be a true update procedure for dynamic windows. -Ed]*

To use WINDOW PICTURE you first define the PICTURE which you want to print. This can be a combination of text and graphics. The PICTURE is pointed to by a variable which is used in the WINDOW PICTURE statement to initialize the automatic update routine. To stop the update or to modify it you can use WINDOW PICTURE #1,0 (if using window #1) and then KILL PICTURE to delete the picture from memory. If you don't remove the pictures from memory, they may use up some of your memory and sooner or later if enough pictures are used, you could run out of memory.

I think the 3rd method is a bit more cumbersome than the first two. I call it the "GET/PUT" method. First, the window contents are drawn for the first time. Then the rectangle containing the contents of the window must be determined exactly. These coordinates are plugged in to an equation given on page 232 of the ZBasic manual. The equation determines the number of bytes to use in a DIM statement for the graphic image used in the GET and PUT statement. The equation is:

```
bytes = 6 + ((y2-y1)+1) * ((x2-x1)+1) * bpp + 7) / 8
```

where x1, y1 are the coordinates of the upper-left-corner of the graphic image on the screen; x2, y2 are the coordinates of the lower-right-corner of the image. The bits-per-pixel, bpp, depends on your Macintosh screen setup. Standard black & white Macintoshes have one bit per pixel, a Macintosh II with sixteen colors is 4 bpp and 256 colors is 8 bpp. Now divide the number of bytes by two for the integer (16 bit) array used in the DIM statement. This is the extent of the difficulty in using the "GET/PUT" method. The DIM statement should be executed only once at the beginning of your program and enough space should be allocated for whatever you plan to update. After the screen is drawn the first time, the GET statement is used to set the array to the screen image. If your array isn't dimensioned big enough you can definitely expect a system bomb here. I would recommend that you only use the "GET/PUT" method when you are not going to change the window contents. The PUT statement is called by the DIALOG event routine whenever the window needs refreshing. (Mac II color users, notice that the PUT statement only refreshes in black & white; the color information is ignored. This is unfortunate. Maybe in a future release of ZBasic with 256 color support?????).

The demo program demonstrates the three methods discussed above. Try opening DAs and dragging a window on top of other windows and closing/opening them. I feel that this program should give you a good idea of how windows can be refreshed. I strongly recommend that most of your refreshing be done with the WINDOW PICTURE statement whenever possible. (NOTE: 'whenever possible' is just about always, except when controls are used on the same screen. Maybe this will get fixed someday too so that WINDOW PICTURE will work when EDIT FIELDs are active).

This has been a much more pleasant program to write as Zedcor has released version 4.01 with a much improved editor. The editor isn't perfect, but if you remember to save your program often enough you can get by without exiting ZBasic. Also, they have added DIALOG(0)=17 for disk inserts. DIALOG(17) returns the disk drive which the disk was inserted. ZBasic is still the best Basic available today and keeps looking better each new release. Zedcor, keep up the good work.

Zedcor and True Basic have announced new versions at the San Francisco Expo. Watch for details on these new products. -Ed

```
' WINDOW UPDATE DEMO
' ©1988 MacTutor©
' By Dave Kelly

WINDOW OFF
COORDINATE WINDOW
DEF MOUSE=-1
DIM Wptr&(3)

APPLE MENU "About Updating..."
MENU 1,0,1,"File"
MENU 1,1,1,"Quit"
EDIT MENU 2
MENU 3,0,1,"Window"
MENU 3,1,1,"Show Window 1"
MENU 3,2,1,"Show Window 2"
MENU 3,3,1,"Show Window 3"
MENU 3,4,1,"Show All Windows"

'Find out monitor size just in case we need it
CALL GETWMGRPORT(WMgrPort&)
PortTop=PEEK WORD(WMgrPort&+8)
PortLeft=PEEK WORD(WMgrPort&+10)
PortBottom=PEEK WORD(WMgrPort&+12)
PortRight=PEEK WORD(WMgrPort&+14)

WINDOW#1,"Window 1",(PortLeft+4,PortTop+42)-(300,300),5
Wptr&(1)=WINDOW(14)
TEXT 3,12
EDIT FIELD 1,"Read MacTutor!",(50,200)-(270,225),1
GOSUB "Set Window 1 Update"

WINDOW#2,"Window 2",(PortLeft+24,PortTop+62)-(320,320),5
Wptr&(2)=WINDOW(14)
'NOW... Create the picture for the first window
PICTURE ON
TEXT 2,24,1
```

```
COLOR=4
PRINT@(3,1)"MacTutor©"
COLOR=7
TEXT 3,12,0
PRINT@(6,4)"The Best in the West!"
PRINT@(5,6)"This is the second window"
PICTURE OFF,Pic2&
PICTURE, Pic2&:' Draw the picture
WINDOW PICTURE #2,Pic2&

WINDOW#3,"Window 3",(PortLeft+44,PortTop+82)-(340,340),5
Wptr&(3)=WINDOW(14)
x1=0:x2=300:y1=0:y2=300
' bytes used = 6 + ((y2-y1)+1) * ((x2-x1+1)* bpp + 7) / 8)
DIM Pic3%(58293) :' space for 32 bits-per-pixel (MAC II)
' NOW... Create the picture for the third window
TEXT 2,24,1
COLOR=4
PRINT@(3,1)"MacTutor©"
COLOR=7
TEXT 3,12,0
PRINT@(6,4)"The Best in the West!"
PRINT@(5,6)"This is the third window"
GET (x1,y1)-(x2,y2),Pic3%(1)
PUT (0,0),Pic3%(1),0

ON DIALOG GOSUB "DialogEvent"
ON MENU GOSUB "MenuEvent"
FLUSHEVENTS
MENU ON:DIALOG ON

"Loop"
GOTO "Loop"
MENU OFF:DIALOG OFF

"DialogEvent"
D=DIALOG(0)
SELECT D
      CASE 3
            WINDOW DIALOG(3)
      CASE 4
            CALL HIDEWINDOW(Wptr&(DIALOG(4)))
      CASE 5,6,7
            IF DIALOG(5)=3 THEN GOSUB"Window3 Update" ELSE
GOSUB "Set Window 1 Update"
END SELECT
RETURN

"MenuEvent"
Menunumber=MENU(0)
Menuitem=MENU(1)
MENU
SELECT Menunumber
      CASE 255
            GOSUB "AppleMenu"
      CASE 1
            GOSUB "FileMenu"
      CASE 3
            GOSUB "WindowMenu"
END SELECT
RETURN

"AppleMenu"
WINDOW 4,"",(100,100)-(400,250),-2
PICTURE ON
TEXT 0,12,0,0
PRINT @(2,2) "(ZBASIC) Window Update Demo V1.0"
PRINT @(10,3)"by"
PRINT @(8,4)"Dave Kelly"
COLOR=6
PRINT @(7,6)"©MacTutor, 1988"
COLOR=7
PRINT @(7,7)"ZBasic version 4.01"

PICTURE OFF,Pic4&
PICTURE ,Pic4&
WINDOW PICTURE #4,Pic4&
MOUSE ON
DO
mous=MOUSE(0)
outsiderect=(MOUSE(1)<0 OR MOUSE(1)>300 OR MOUSE(2)<0 OR
MOUSE(2)> 150)
IF MOUSE(1)=0 AND MOUSE(2)=0 THEN "Stop"
UNTIL mous<>0 AND NOT (outsiderect)
MOUSE OFF
KILL PICTURE Pic4&:' Delete the picture from memory
DIALOG ON
WINDOW CLOSE 4
DIALOG OFF
RETURN

"Window3 Update"
ActiveWindow=WINDOW(0)
WINDOW OUTPUT 3
PUT (0,0),Pic3%(1),0
WINDOW OUTPUT ActiveWindow
RETURN

"Set Window 1 Update"
ActiveWindow=WINDOW(0)
WINDOW OUTPUT 1
'NOW... Create the picture for the first window
      KILL PICTURE Pic1&
      PICTURE ON
      TEXT 2,24,1
      COLOR=4
      PRINT@(3,1)"MacTutor©"
      COLOR=7
      TEXT 3,12,0,0
      PRINT@(6,4)"The Best in the West!"
      PRINT@(5,6)"This is the first window"
      Editvariable$=EDIT$(1)
      PRINT@(5,8)"The edit field = ";Editvariable$
      PICTURE OFF,Pic1&
PRINT@(5,8);SPC(100)
PICTURE, Pic1&:' Draw the picture
WINDOW OUTPUT ActiveWindow
RETURN

"FileMenu"
IF Menuitem=1 THEN "Stop"
RETURN

"WindowMenu"
SELECT Menuitem
      CASE 1
            CALL SHOWWINDOW(Wptr&(1)):WINDOW 1
      CASE 2
            CALL SHOWWINDOW(Wptr&(2)):WINDOW 2
      CASE 3
            CALL SHOWWINDOW(Wptr&(3)):WINDOW 3
      CASE 4
            FOR i=1 TO 3
                  CALL SHOWWINDOW(Wptr&(i))
                  WINDOW i
            NEXT i
END SELECT
RETURN

"Stop"
WINDOW PICTURE #1,0:'Don't update the first window anymore
KILL PICTURE Pic1&:' Delete the picture from memory
WINDOW PICTURE #2,0:'Don't update the second window anymore
KILL PICTURE Pic2&:' Delete the picture from memory
END
```

© The Definitive MacTutor, Vol. 4

Basic School

Color Mixer Creates Custom Colors

Dave Kelly
MadTutor Editorial Board
Ontario, California
MacTutor Vol. 4 No. 4

ZBasic Does Mac II Color!

Yes, that's right! The new ZBasic version 4.01 now supports the Mac II's full 256 colors. There is no need to restrict yourself to the 8 colors of the old Quickdraw.

To use the 256 color mode, ZBasic now provides a LONG COLOR statement. The syntax is: **LONG COLOR** blue%, green%, red%, [background]. LONG COLOR is used to set the colors and grey levels for subsequent graphic commands used on the Macintosh II. The command is ignored by other Macs.

The values of blue%, green% and red% may range from 0 to 65535. They may also be represented as long integers (65535 as a long integer is -32767 as a regular integer). The colors are used in RGBColor calls to the Mac II ROM where 0 is the darkest and 65535 is the lightest.

The background variable is used as a flag to indicate if the color should be set in the foreground or in the background. A background value of 0 causes the background color to be set to the blue%, green%, red% color indicated. Subsequent LONG COLOR statements that have no background variable or background is set to other than 0 will set the foreground color. The background color will be used the next time it is needed, when the screen gets updated or is cleared with CLS.

Inside Macintosh Volume V has the new color information we need to understand how the Mac II gets its color and how to use color in your applications. Before using color you should read the User Interface chapter of Inside Macintosh Vol. V. One of the most important thing you should try do is leave the selection and choices of color in the hands of the user. The study of color and how it affects people is not well defined. One of my co-workers is color-blind and has a totally different way of looking at color than I do. If color is not used carefully it could hinder instead of help to enhance your software.

Traditionally (and on that other computer) color has been used to help distinguish different things on the screen and show relationships between items on the screen. Some colors have special significance. For example: red means stop, green means go, yellow means caution. Colors can enhance in this way, however, remember that after the user has seen the same color combinations several dozen times, the color becomes commonplace and he might still make mistakes that the color was supposed to help him avoid. In fact, the bright colors used to warn a user may actually attract him to select that color.

Inside Mac encourages you to start out the design of your program in black and white and add color as a supplement to provide extra for people who have color capability. Color use should be limited to the content area of your windows. Let the user decide what color his menus, borders, controls etc. should be. The Colorizer application and CDEV let the user select his own colors for these items. If the user selects colors for somethings then your application comes along and changes part of these colors, there might be some pretty strange color combinations left over. Many suggestions are available in Inside Mac which you should know so that your applications will be consistent and won't conflict with the user interface.

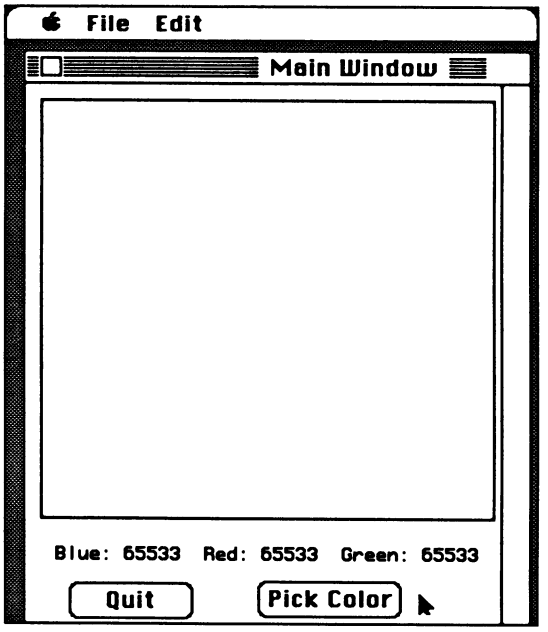


Fig. 1 Mixing colors, in this case white!

The blue%, green%, and red% that you specify in the LONG COLOR is used by the Color Manager to return the pixel value that best represents that color. The Color Manager searches through the color table for the RGB color that most closely matched the desired color. Color Quickdraw then places that color on the screen. The way that each pixel is assigned a color to display is by way of the Color Look-Up Table (CLUT). The video card takes each pixel and compares it to the CLUT to determine which RGB value that pixel represents. The RGB value is split off into three separate electrical signals (red, green, and blue) to the video monitor. By the way, when red=green=blue you get shades of gray. By using the Palette Manager (if you could from Basic) you could change the drawing environment to use a different palette.

Color Selection

Color Selection routines have been provided to the Macintosh II by Apple via the Color Picker routines. Here is the problem: ZBasic doesn't support all the new calls for the Mac II yet. Zedcor has promised to send me format and structure information to add more calls to ZBasic as required, but Andy Gariepy has been to busy to send it to me. So... that being the case I have no choice but to write my own color selection routines. Also the Color Picker is PACK_12 in system 4.2. I'm not sure it would be that easy to call a PACKage from ZBasic without it being defined by the language itself. Maybe we will have to wait for Zedcor to implement it. In the mean time I have made my own "quick and dirty" version of a "Color Picker".

If you think about it, the Apple Color Picker routine really does do a lot! It allows you to type in the red, green, blue colors, but also will adjust these colors when you set the hue, saturation, and brightness. Proper adjustment of the red, green, and blue will result in the same hues, saturation, and brightness, but it just isn't as convenient to setup.

The color wheel concept is interesting too. You can learn a lot about how colors can be mixed by watching the values of red, green, and blue while moving the mouse around the color wheel. Move more towards blue and watch how the numbers change! Amazing! The color wheel really is a good way to allow you to select colors. The only other good approach is to allow the user to pick from a color palette. The problem is how to decide which colors are defined in the palette.

My example suggests a simple way that colors could be selected by the user. By choosing the Pick Colors menu or button you are given a set of 3 scroll bars to adjust the amount of blue, green, and red. You can mix the amount of each color and see the effect of mixing each. This might even be useful for showing young children the effects of mixing colors. If you like the color then click on OK or else cancel to return to the original color setting.

The problem with my Color Selection demo is that because of the way ZBasic does event trapping it is difficult to make the routines be stand alone. My color mixer program provided here must become an integral part of the application because of the way that ZBasic "automatically" generates the event loop. I tried using separate ON DIALOG GOSUB traps, but it somehow didn't work. I could only get one of the ON DIALOG statements to take effect.

There are a few problems that have shown up with my early version of ZBasic 4.01. First, the scroll bars don't scroll properly when using the max and min values that are given in the ZBasic manual. If +32767 is used and you scroll to the right (or down) by clicking on the right (or down) arrow, when the bar reaches the max it will jump to the minimum! Also if you change the max value to between 32100 and 32766 and click in the scroll bar between the arrow and box, when the max is reached it also jumps down to the minimum. UGH! And... if -32768 is used for the minimum the scroll bar returns a 0 when the box is set at the minimum. -32767 works fine. I am forced to adjust these values in the program, but it won't matter much because the selected color will likely be the same anyway because we only have 256 possible.

Hierarchical Menu Notes

Using Hierarchical menus isn't so simple with ZBasic. Hierarchical menus are stored as resources of the type MENU just as other menus may be stored. However, to use MENU resources with ZBasic AND have ZBasic be able to use them, you have to set up your own event loop using GETNEXTEVENT. Then you have to handle each event with the standard Macintosh way of programming and can't use ZBasic's event trapping routines (they get in the way). What we need is a way to use MENU, WINDOW, CONTROLs and other resources from ZBasic which can be invoked with ZBasic statements (which are integrated in with the ZBasic "automatic" event loop. It would also be useful to be able to automatically install a resource into an application when ZBasic compiles as an application. Maybe Zedcor can put these things on their "THINGS TO DO" list for version 4.02?

A submenu is associated with a menu item by using 2 fields of the MenuInfo data structure. When the itemCmd field is set to hex $1B, the itemMark field should contain the menuID (in hex) of the associated hierarchical menu. These values can be created/modified with ResEdit 1.1B3.

By using Menu Manager calls (pg. E-181 in ZBasic manual) you can find out which menu is selected in the GETNEXTEVENT loop as you would with regular type menus. More on this in a future column.

```
'Professor Mac's ZBasic Color Mixer
'©MacTutor 1988
'By Dave Kelly

WINDOW OFF
COORDINATE WINDOW
DIM Colorbox%(3)
red&=0
blue&=0
green&=0

DEF MOUSE=-1
false=0:true=NOT(false)
MENU 1,0,1,"File"
MENU 1,1,1,"Select Color"
MENU 1,2,1,"Quit"
EDIT MENU 2

'Find out screen size.
CALL GETWMGRPORT(WMgrPort&)
PortTop=PEEK WORD(WMgrPort&+8)
PortLeft=PEEK WORD(WMgrPort&+10)
PortBottom=PEEK WORD(WMgrPort&+12)
PortRight=PEEK WORD(WMgrPort&+14)

WINDOW 1,"Main Window",(10,44)-(PortRight-4,PortBottom-4)
TEXT ,,,0
BUTTON #6,1,"Quit",(20,WINDOW(3)-50)-(85,
        WINDOW(3)-30),1
BUTTON #7,1,"Pick Color",(120,WINDOW(3)-50)-(195,
        WINDOW(3)-30),1
```

```
CALL SETRECT(Colorbox%(0),10,10,WINDOW(2)-10,
        WINDOW(3)-215)
PICTURE ON
LONG COLOR blue&,green&,red&
CALL PAINTRECT(Colorbox%(0))
CALL MOVETO(20,WINDOW(3)-100)
COLOR=7
PRINT "Blue:";blue&;" Red:";red&;" Green:";green&;SPACE$(10)
PICTURE OFF,Pic1&
PICTURE, Pic1&
WINDOW PICTURE #1,Pic1&

ON DIALOG GOSUB "Event"
ON MENU GOSUB "MenuEvent"
DIALOG ON:MENU ON
"Loop"
GOTO "Loop"
DIALOG STOP:MENU STOP

"MenuEvent"
Menunumber=MENU(0)
Menuitem=MENU(1)
MENU
IF Menunumber<>1 THEN RETURN
SELECT Menuitem
    CASE 1
        GOSUB "ColorPick"
    CASE 2
        GOSUB "Quit"
END SELECT
RETURN

"Quit"
WINDOW PICTURE #1,0
KILL PICTURE Pic1&
END

"ColorPick":'A Color Selection routine
done=false
min%=-32767:'Manual says that -32768 is Min.
max%=32766:'Manual says that 32767 is Max.
currentred&=red&
currentblue&=blue&
currentgreen&=green&
WINDOW 5,"Color Chooser",(PortLeft+10,PortTop+30)-
    (PortRight-10, PortBottom-10),-2
SCROLL BUTTON #1,currentblue&-32767,min%,max%,100,(20,
    WINDOW(3)-200)-(WINDOW(2)-20,WINDOW(3)-200+16),0
SCROLL BUTTON #2,currentgreen&-32767, min%, max%, 100,
    (20, WINDOW(3)-150)-(WINDOW(2)-20, WINDOW(3)-150+16),0
SCROLL BUTTON #3,currentred&-32767, min%, max%, 100, (20,
    WINDOW(3)-100)-(WINDOW(2)-20,WINDOW(3)-100+16),0
BUTTON #4,1,"OK",(20,WINDOW(3)-50)-(85,
    WINDOW(3)-30),1
BUTTON #5,1,"Cancel",(120,WINDOW(3)-50)-(185,
        WINDOW(3)-30),1
TEXT 0,12,0,0
CALL MOVETO(WINDOW(2)/2,WINDOW(3)-200+30)
String$="Blue: "+STR$(currentblue&)
COLOR=4
CALL DRAWSTRING(String$)
CALL MOVETO(WINDOW(2)/2,WINDOW(3)-150+30)
String$="Green: "+STR$(currentgreen&)
COLOR=2
CALL DRAWSTRING(String$)
CALL MOVETO(WINDOW(2)/2,WINDOW(3)-100+30)
String$="Red: "+STR$(currentred&)
COLOR=6
CALL DRAWSTRING(String$)
GOSUB "UpdateColorBox"
FLUSHEVENTS
RETURN

"ButtonEvent"
Buttonnumber=DIALOG(1)
Buttonvalue&=BUTTON(Buttonnumber)+32767
SELECT Buttonnumber
    CASE 1
        GOSUB "Button1":'set blue
    CASE 2
        GOSUB "Button2":'set green
    CASE 3
        GOSUB "Button3":'set red
    CASE 4
        GOSUB "Button4":' OK button was pressed
    CASE 5
        GOSUB "Button5":' Cancel button was pressed
    CASE 6
        GOSUB "Quit"
    CASE 7
        GOSUB "ColorPick"
END SELECT
RETURN

"UpdateColorBox"
LONG COLOR currentblue&,currentgreen&,currentred&
CALL SETRECT(Colorbox%(0),10,10,WINDOW(2)-10,
        WINDOW(3)-215)
CALL PAINTRECT(Colorbox%(0))
RETURN

"Button1":'blue scroll bar
currentblue&=Buttonvalue&
CALL MOVETO(WINDOW(2)/2,WINDOW(3)-200+30)
String$="Blue: "+STR$(currentblue&)+SPACE$(10)
COLOR=4:'set text color to blue
CALL DRAWSTRING(String$)
GOSUB "UpdateColorBox"
RETURN

"Button2":'green scroll bar
currentgreen&=Buttonvalue&
CALL MOVETO(WINDOW(2)/2,WINDOW(3)-150+30)
String$="Green: "+STR$(currentgreen&)+SPACE$(10)
COLOR=2:'set text color to green
CALL DRAWSTRING(String$)
GOSUB "UpdateColorBox"
RETURN

"Button3":'red scroll bar
currentred&=Buttonvalue&
CALL MOVETO(WINDOW(2)/2,WINDOW(3)-100+30)
String$="Red: "+STR$(currentred&)+SPACE$(10)
COLOR=6:'set text color to red
CALL DRAWSTRING(String$)
GOSUB "UpdateColorBox"
RETURN

"Button4":'OK button
red&=currentred&
blue&=currentblue&
green&=currentgreen&
GOSUB "Closestuff"
RETURN

"Button5":'Cancel button
GOSUB "Closestuff"
RETURN

"Closestuff"
FOR J=1 TO 5
    BUTTON CLOSE J
NEXT J
WINDOW CLOSE 5
PICTURE ON
LONG COLOR blue&,green&,red&
CALL PAINTRECT(Colorbox%(0))
CALL MOVETO(20,WINDOW(3)-100)
```

```
COLOR=7
PRINT "Blue:";blue&;" Red:";red&;" Green:";green&;SPACE$(10)
PICTURE OFF,Pic1&
PICTURE, Pic1&
WINDOW PICTURE #1,Pic1&
RETURN

"Event"
D=DIALOG(0)
SELECT D
    CASE 1
        GOSUB "ButtonEvent"
    CASE 4
        GOSUB "Quit":'if close box selected
END SELECT
RETURN
```

Hypercard Tidbits

At MacWorld Expo in San Francisco someone asked if we knew a way to effectively mask out the command-option keys in HyperCard. The following information was given in WINOID issue #1, a newsletter for the Apple HyperCard Users Group. I hope that this will answer your question.

How can I hide buttons from command-option key peek?

When you don't want people peeking at your buttons, it is necessary to trap the command and option keys. While there are several ways of doing this, here is one that is simple and works about 99% of the time.

The basic idea is to take the user to another card when the command and option keys are both pressed. This can be a "no peeking" card or a rules card or some other kind of card. When the user releases both keys, you then return to the card the user was viewing before trying to peek. Following are the two scripts that accomplish this. Put the top one in your stack script and the other in the "no peeking" card script.

For the stack script:

```
On Idle
  if the commandkey is down and the optionkey is down then
      push this card
      go card "no peeking"
  end if
end Idle
```

For the "no peeking" card script:

```
On Idle
  if the commandkey is up and the optionkey is up then
      pop card
  end if
End Idle
```

© The Definitive MacTutor, Vol. 4

Basic School
Quick & Dirty Data Conversion

Dave Kelly
MacTutor Editorial Board
MacTutor Vol. 4 No. 5

FILE IMPORT/EXPORT with BASIC

Over the years, one of the most beneficial uses for Basic is the ability to quickly write programs to import/export data to and from data bases. This also provides a "quick and dirty" way to convert data or suppress unwanted characters. I'd like to explain/complain about the process and some of the problems that I have encountered.

My first major data conversion came a couple of years ago when I converted 30,000 names in a mail list from my Apple // e to the Macintosh. Yes, this was quite a feat, mostly because of the bottle neck which the dinky 143K floppy drives allowed on the Apple //. And you thought the Mac 400K drives were small! Anyway, on the Apple //e I was using DB MASTER, a now orphaned product, with some small support still available from a consulting firm in Northern California. There may be some of you that are familiar with DB MASTER (sometimes it was referred to as DB MONSTER). DB MASTER was great for mail lists and still would be today except for it's slow speed (that's partly the Apple //s fault).

The process went something like this: First the data needed to be converted from DB MASTER file format to ASCII so that I could direct connect the Mac and the //e together. DB MASTER utilities includes an export utility which would convert files to either DIF or a "One Field per Line" format. The utilities converted my mail list from my Sider hard disk to about 50 143K floppy disks (UGH!). Next the ASCII needed to be converted from Apple // ASCII to "the rest of the world" ASCII format. ASCII EXPRESS telecommunications software provides a conversion utility for doing this (If they didn't I'd have to use my own Basic program to convert it myself. Next each disk was converted via direct null modem connection to the Mac. Whew!

To make a long story (and many hours) short, the mail list was now on the Mac, but somehow during the process some control characters were intermixed in the files. Omnis 3 would not read the files with these control characters so it was BASIC to the rescue. There are 128 standard ASCII characters of which the first 31 characters are control characters, traditionally used for teletype terminals. The problem was easily solved by writing a Basic routine to read in the file, filter out the bad characters and rewrite the file. Compile the program to speed it up!

```
Control Character Filter
' ©MacTutor, 1988
' By Dave Kelly
' MSBASIC

x$=FILES$(1,"TEXT")
IF x$="" THEN END
OPEN x$ FOR INPUT AS #1
OPEN "Filter Output" FOR OUTPUT AS #2
WHILE NOT EOF(1)
    Char$=INPUT$(1,1)
    IF ASC(Char$)>=32 THEN PRINT #2,Char$;
    IF Char$=CHR$(13) THEN PRINT #2,Char$;
WEND
CLOSE #1
CLOSE #2
END
```

That's ok for simple filtering. Things may need to be more complex if the original file contains unexpected characters in it. Recently, I came across the need to convert data stored in a FileMaker Plus file in order to do some simple, but unusual calculations. As drafting is done for test equipment the progress is logged into a FileMaker Plus file. Each drawing number is entered with its own charge number and the date is entered indicating the start or completion of stages of the drawing process. Occasionally the boss (who is always right because he is the boss) wants a drawing status report according to charge number (there are multiple drawings per charge number). FileMaker Plus will give the total number of drawings but only after searching for each of the charge numbers and dates individually. BASIC to the rescue.

The Drawing Count program was written to count progress of each of the charge numbers. A file is opened for input and another file for output. Each field was previously exported to a BASIC formatted file and is read by the Basic program in the same order which it was written. The charge number is saved, but each of the date fields is replaced by a flag (0 or 1) which indicated if there was anything in the date field. These 0s and 1s are added together to get the totals after the data is sorted. The sort routine could have been beefed up but, for a "quick and dirty" program, it does the job. The program was compiled with the MS Basic compiler which provided adequate speed results.

If you are wondering why ZBasic wasn't used here, so am I. Actually, I ran into a small problem which appears to be a bug with ZBasic 4.01 while reading the FileMaker Plus exported text files. Although ZBasic's INPUT# statement is supposed to read data from the file until a carriage return, <COMMA>, End-Of-File, or 255 characters are encountered, when ZBasic saw a comma followed by a carriage return, the null between the comma and carriage return was skipped and the next field (from the next record) was read in. The data looks like this:

```
"ITEM1","ITEM2","ITEM3",
"ITEM4","ITEM5","ITEM6","ITEM7"
```

The null field after item3 was skipped and ITEM4 is read in. This is how data is formatted when dumped in comma delimited format as with FileMaker Plus. Sorry, Zedcor.

```
COORDINATE WINDOW
DEF OPEN "TEXTDAVE"
BREAK ON
filename$=FILES$(1,"TEXT",,vol%)
IF filename$="" THEN END
OPEN "I",#1,filename$,,vol%
INPUT #1,A$,B$,C$,D$
PRINT @(1,10)A$,B$,C$,D$
CLOSE 1
```

Another file conversion problem to deal with involves the conversion of text which has been dumped from text fields. FileMaker Plus, for example, inserts an end of line mark at the end of each line when it exports data to a file. When the data is read into another program which doesn't use the end of line character to mark the end of lines (because the text is word wrapped and reformattable). If you read the FileMaker text field into Double Helix II for example, the end of line character shows up in the text field as an undefined character (a box). The filter method explained above can find these characters if you know what character it is.

Fortunately life is much easier for converting data these days than it was even five years ago. Now I use file formats like DIF and SYLK whenever I can when converting between programs that support both formats. The transfer goes a lot less painfully when the format is well defined as the DIF format is. For those not familiar with DIF, it is called the "Data Interchange Format" and was originally designed to by Software Arts (are they still around?) to transfer data from Visicalc (the father of all spreadsheet programs). The file is completely ASCII but has header information added to help maintain data integrity. (This saved me a few times). This means that the file will undoubtedly be larger than without DIF formatting so be sure that you only use it for file transport rather than data storage especially if your disk free space is critical. SYLK is Microsoft's format. Most Microsoft products and a few data base programs such as Omnis 3 and FileMaker Plus support SYLK format. The best way to figure out how these work is to analyze some sample data from a program that exports in one of these formats.

```
' Drawing Count Program
' Converts Drawing Log Data created by FileMaker Plus
' and computes total number of drawings under each date
' By Dave Kelly
' ©1988
'MS BASIC

DIM Charge$(1000),Date1(1000),Date2(1000),Date3(1000),
Date4(1000)
DIM tCharge$(20),tDate1(20),tDate2(20),tDate3(20),tDate4(20)
filename$=FILES$(1,"TEXT")
IF filename$="" THEN END
OPEN "I",1,filename$
OPEN "O",2,"Drawing Count.DATA"
PRINT
count=0
CALL MOVETO(10,30)
PRINT "Now Reading in Records..."
WHILE NOT EOF(1)
    count=count+1
    CALL MOVETO (10,50)
    PRINT count
    Charge$(count)=""
    i$=""
    A$=""
    C$=""
    S$=""
    INPUT#1,Charge$(count),i$,A$,C$,S$
    IF LEN(Charge$(count))>0 THEN
        IF RIGHT$(Charge$(count),1)=")" THEN
       Charge$(count)=MID$(Charge$(count),LEN(Charge$(count))-
4,4)
        ELSE
            Charge$(count)=RIGHT$(Charge$(count),4)
        END IF
    END IF
    IF LEN(i$)>0 THEN Date1(count)=1 ELSE Date1(count)=0
    IF LEN(A$)>0 THEN Date2(count)=1 ELSE Date2(count)=0
    IF LEN(C$)>0 THEN Date3(count)=1 ELSE Date3(count)=0
    IF LEN(S$)>0 THEN Date4(count)=1 ELSE Date4(count)=0
WEND
CALL MOVETO(10,70)
PRINT "Now Sorting Data..."
CALL Sort(count,Charge$(),Date1(),Date2(),Date3(),Date4())
k=0
FOR j=1 TO count
IF tCharge$(k)<>Charge$(j) THEN k=k+1
tCharge$(k)=Charge$(j)
tDate1(k)=tDate1(k)+Date1(j)
tDate2(k)=tDate2(k)+Date2(j)
tDate3(k)=tDate3(k)+Date3(j)
tDate4(k)=tDate4(k)+Date4(j)
NEXT j
CALL MOVETO(10,90)
PRINT "Now Writing to File..."
PRINT#2,"Charge#    Date1    Date2    Date3    Date4"
FOR i=1 TO k
tDate1(i)=tDate1(i)-tDate2(i)
tDate2(i)=tDate2(i)-tDate3(i)
tDate3(i)=tDate3(i)-tDate4(i)
PRINT #2,tCharge$(i),tDate1(i),tDate2(i),tDate3(i),tDate4(i)
PRINT tCharge$(i),tDate1(i),tDate2(i),tDate3(i),tDate4(i)
NEXT i
CLOSE #1
CLOSE #2
PRINT"Click to continue..."
WHILE MOUSE(0)<>1:WEND
END

SUB Sort(count,Charge$(),Date1(),Date2(),Date3(),Date4())
STATIC
    flips=1
    WHILE flips
        flips=0
        FOR i=2 TO count
            IF Charge$(i-1)> Charge$(i) THEN
                flips=1
                SWAP Charge$(i-1),Charge$(i)
                SWAP Date1(i-1),Date1(i)
                SWAP Date2(i-1),Date2(i)
                SWAP Date3(i-1),Date3(i)
                SWAP Date4(i-1),Date4(i)
            END IF
        NEXT
    WEND
END SUB
```

HYPER-CORNER

The new update disk for HyperCard version 1.1 contains a read me stack with three new buttons pertaining to importing and exporting data. If you don't have the new update you can get it from your Apple dealer. Version 1.1 is supposed to fix some bugs that "bugged" some of us.

The read me stack gives directions for using the new buttons, but I'll try to explain them a little bit here. To try them out, create

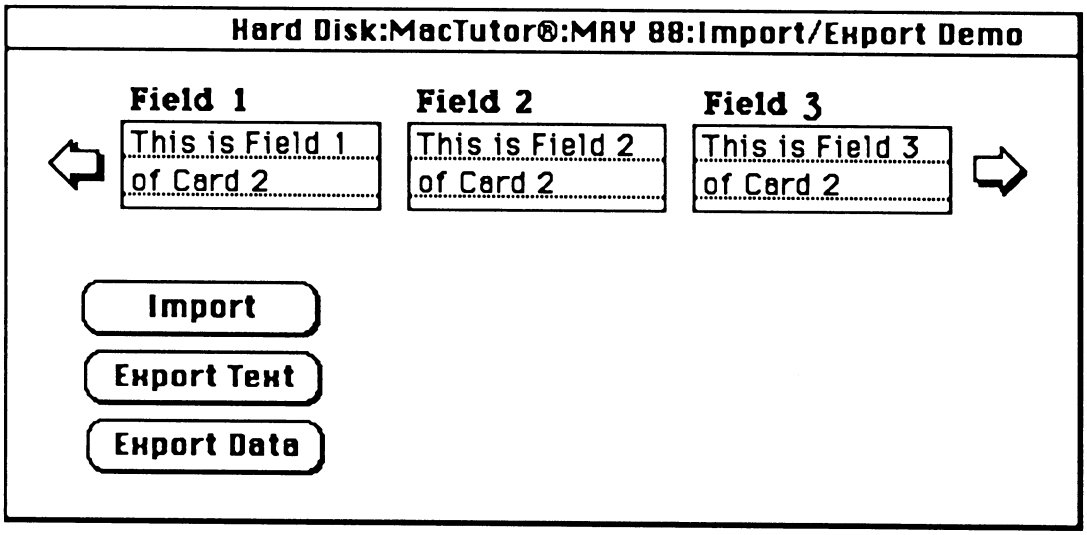


Fig. 1 Hyper Dialog

a new stack with a few fields in it such as the one I created; see fig. 1.

HyperCard Import/Export

The arrow keys were added from the Button Ideas stack to make it easy to check out the data, but you will need to remember the command 1,2,3,4 keys that move you from card to card or you may use the arrow (or option-arrow) keys to move from field to field.

The most useful of the buttons is the Export Data button. It allows you to select the background fields you wish to export and the delimiters separating the fields and records. Most database programs will read the tab delimited records (with carriage return between each record). NO, the export button will not export card fields, only background fields. That is understandable since the output needs to be consistent.

I found that the best way to see how the buttons work is to try them out. Clicking on the Export Data (or Export Text) button

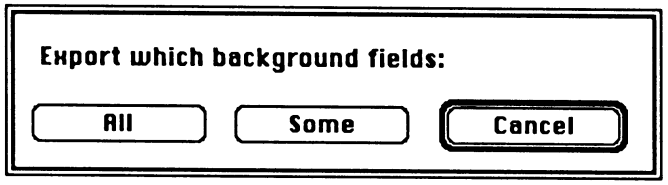


Fig. 2 Export Dialog

will produce this dialog which will let you select the background field you desire to output. (See figure 2)

Selecting the Fields for Export

After selecting the All button or after selecting which of the background fields you want to export, you specify the file name to export to. A dialog comes up to ask you for the name of the text file to export in figure 3.

Selecting a Filename for Export

There is an XCMD around someplace for calling the Standard File Dialog which would be better to substitute for this dialog. The problem with XCMD routines for general use is that not everyone has them. It would be best to set up your own custom library of XCMDs to use in your own stacks.

I ran into no difficulty getting my data to export to a file. You can examine your data with a word processor to see how the text file looks when the data has been exported. I like using MS Word for examining this kind of file because you can view all the tabs and carriage returns using the Show ¶ menu item.

The Import Button works well too, except that you can't import data into

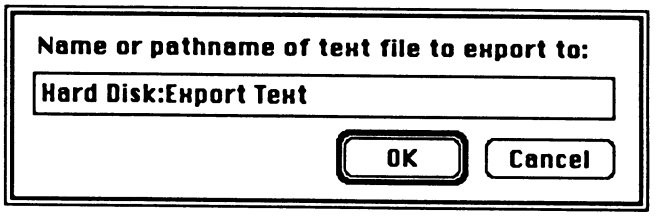
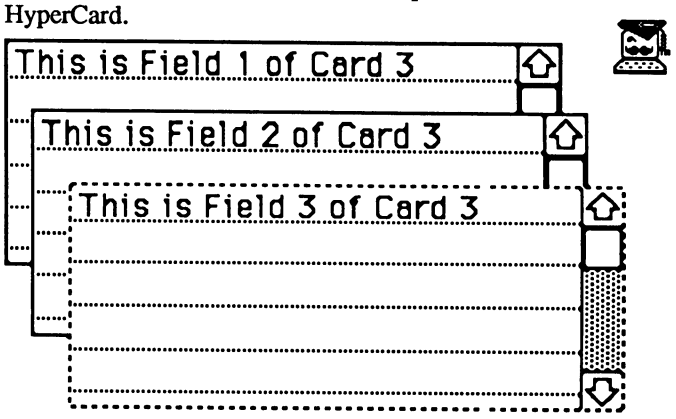


Fig. 3 Editable Dialog

existing background fields. It creates the background fields on the fly. Perhaps someone out there could create a button script that will allow you to specify which field you would like to be the first, second and so on field to be read in. AT LEAST, now these buttons are included with HyperCard. At first, it may appear that the import didn't work, but if you read the instructions in the read me stack, it explains that the new background fields are created one on top of another. To see the fields you will have to peel them apart. (see figure below)

Imported Fields

Ideally, the importing will be done first thing before the stack layout is designed if possible. Importing/Exporting of text with HyperCard has been one of the weak links. With all the interest in HyperCard these days, there should be more XCMD routines and tools we can use to improve the usefulness of HyperCard.



Basic School
Hierarchical Menus & Color Notes

Dave Kelly
MacTutor Editorial Board
MacTutor Vol. 4 No. 6

Hierarchical Menus in ZBasic

Hierarchical menus (also known as submenus) are possible in ZBasic with a small amount of sweat. Hierarchical menu items have the small filled black triangle pointing to the right in many of the newer application programs that are coming out now. This month I will show you how you can implement them in your own ZBasic source code.

There are a few standard guidelines for using submenus which you can find in *The Macintosh User Interface Guidelines* found in Vol. 5 of *Inside Macintosh*. In the pull-down menu structure of the Macintosh, it was originally intended that the user could quickly scan through all of the menus and know the function of all the menu items. Therefore, it is strongly recommended that submenus only be used for lists of related items. You should try to only use one level of hierarchical menus although it is possible to go several levels.

The concepts of submenu programming is fairly simple and easy to implement. However, implementing submenus in ZBasic without some understanding of how things work will probably frustrate some of the average Basic programmers out there. The first thing we have to do is throw away the ZBasic event loop and event handling statements and set up our own GETNEXTEVENT loop. This sounds difficult to some of you, but actually it solves some of the problems inherent in ZBasic that I've been pulling my hair out over for the past year or so. In this month's sample program I even went a step further by defining some of the menus and the main window using MENU and WIND resources. You can refer back to my GETNEXTEVENT ZBasic shell program in September 1987 MacTutor. This will give you the program shell to start with.

First create the window resource. Any window type will do for this example. I chose the rDocProc type window which has a ProcID number 16. To create the WIND resource, open ResEdit and select New from the File menu and type in the WIND resource type. Then select New to create a resource and type the following parameters into your window (as shown in the ResEdit Window in figure 2):

```
boundsRect 54 78 280325
procID  16
visible True
goAway  False
refCon  0
title   Hierarchical Menu Sample
```

Change the WIND ID to 29414 (or any other unique number) by selecting Get Info from the File menu. Now the WIND resource is created. This window will be used by the program to output confirmation that everything works.

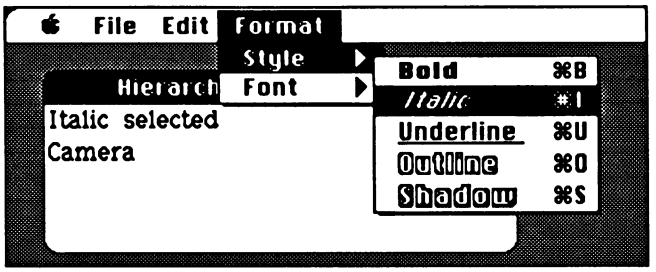


Fig. 1 Our hierarchical menu example in ZBasic

The menus are just as simple to create with ResEdit as the WIND resource was. The parameters for each of the menus is shown in the window dumped from ResEdit which are included here. Two menus are created with ZBasic Menu statements which are not included as resources. Actually, it would probably be better to use all toolbox calls instead of ZBasic Menu statements because we could be in complete control of all the menus. MENU id 255 is the Apple menu created with the APPLE MENU statement in ZBasic, but it is very automatic and you have to have desk accessories whether you like it or not. If you created your own, you could exclude any part you didn't want, but you would have to control every little thing that happens. MENU ID 2 is created with the ZBasic EDIT MENU statement. This is the easiest way to make an Edit menu. MENU ID 134 is the File menu. It contains only one menu item, "Quit".

MENU ID 135 is the Format menu which is the main menu which the submenus will be attached. The Format menu contains two items, "Style" and "Font" which will each have submenus. Now comes the fun part. To attach a menu we first set the "key equiv" for the menu item to $1B. This will put the black filled triangle in the menu. A $1B is ASCII character 27, the ESCAPE key. If you have an Extended keyboard you will have no trouble producing this character by typing "esc". Typing CTRL-[will also produce the ESC character. However, I haven't found a good way to produce ESC using the MacPlus or earlier keyboards; anybody know how??. Next the ID of the submenu is put in the mark Char field. Actually, you enter the character which has the ASCII value of the ID of the submenu. In the Style menu the ASCII value of "à" is 136. This character is entered by using the "option-`" then "a". The Font menu has submenu with ID=137 so the mark Char is "â" which is entered by typing "option-i" and the "a". You should examine the ResEdit windows shown here if you need more help understanding this.

After the MENU and WIND resources are complete, the program can be compiled. You will need to compile the program

© The Definitive MacTutor, Vol. 4

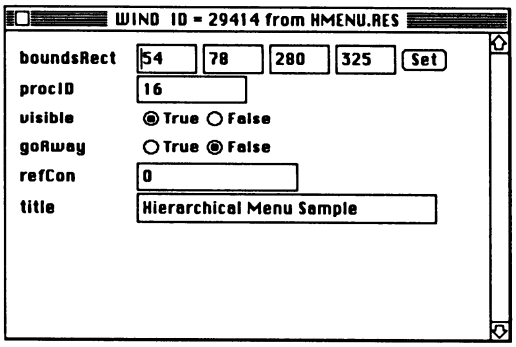


Fig. 2 ResEdit version of the Window Resource

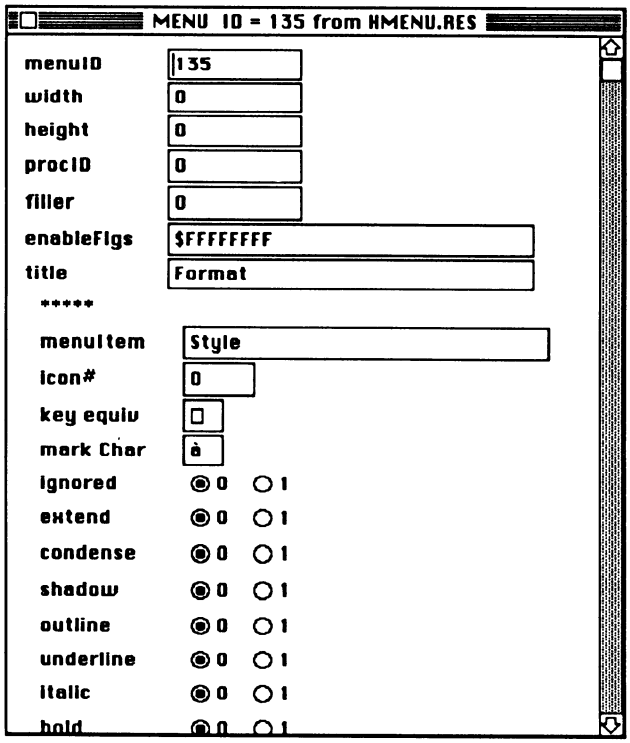


Fig. 3 Menu Resource

as an application and install the MENU and WIND resources before running the program or unexpected results will occur. Remember that the program needs to have the resources in order to work properly. Be sure that the resources are installed by running ResEdit immediately after compiling. It would be nice if ZBasic would install a resource file for you as in "real" development languages. Maybe they will add this capability someday.

The program really doesn't do anything except respond to the menu and print to the window which menu was selected. There are a few left over functions that were originally part of the GETNEXTEVENT shell program. The shell could be expanded to work with other applications you may like to do.

An alternative to using ResEdit is to create your resources with RMaker, as shown in the program listing. Once the resources are compiled, ResEdit can be used to move the resources from the RMaker output file into the ZBasic compiled file.

Sorry State of Mac Basics

Thus far, the Mac II has been out over a year and as of the date of this writing, there is still no Basic released which fully supports all the new ROM capabilities. Even the method I've shown here is a work around because of the way that ResEdit has to be used. It still holds true, however, that ZBasic has the most features and is faster than any of the competitors. Microsoft still has no new updates of MS Basic. And with all the bugs in the compiler, Absoft doesn't want to add any new features till the old ones are fixed. Somebody is walking around with blinders on and doesn't even know it. From the response I've seen, most people are dissatisfied with all the Basic products that are currently released. There are still some possibilities coming up... True Basic 2.01 was released this week. We will be taking a good hard look at the new True Basic in upcoming issues. The grapevine says that Microsoft is readying a new release for sometime? All I can say is that I hope they are listening to all of us and what we are screaming for. *[Note that True Basic 2.0 was found to be defective shortly after shipment, and the company has upgraded immediately to 2.01. It seems that a memory management bug exists in 2.0 that is actually hidden by the presence of the Macsbug debugger, which nearly all their beta testors were using. When they tried the product without Macsbug installed, a problem was found that seems to prevent True Basic from picking up on mouse and keyboard events properly during program execution. We discovered it when we noticed the demos would not stop without powering down the Mac. So be sure to check that your upgrade disk is version 2.01, and a word of warning to other developers, that the presence of a debugger may be altering or hiding potential memory manager bugs. -Ed]*

By the way, my daughter helped to discover a undocumented feature of ZBasic. She was quietly invading my Mac II (when I wasn't looking) while I had ZBasic running and happened to press the arrow keys. Well... it turns out that the down arrow key lists the next sequential program line. The up arrow key prints the previous line. Right arrow lists the last line and Left arrow lists the first line. Interesting, but I prefer to do all my editing using an editor. I'd like to say I like using the built in editor the best, but it still doesn't quite work right.

True Basic 2.01 & Color Models

One of the neat features of the new True Basic 2.01 is that it supports Macintosh II color including palette animation. In fact, the palette animation article we did last month in MacTutor can be done in just a few lines in True Basic! And a neat Mac touch they have added is that the output window for True Basic is saved and refreshed and can be written to disk as a PICT2 resource, unlike MS Basic, which "forgets" how to update the output window after a program runs. To help us get up to speed on color processing on the Mac, here are some notes on color types.

Along with color and the Mac II there also comes new terminology. I'm sure there are many of you that either haven't wanted to or have had no opportunity till now to know about the various color models that exist. Most of us have heard of RGB, some for the first time when they bought their Mac II with RGB monitor. RGB is yet only one of several color models being used

in the computer graphics industry. Some other color models include: CMY, YIQ, LAB, HSL, HSV, and HSI Color Models.

I will attempt to explain them from the few things that I have read on the subject. Each model has its advantages and disadvantages. In general the color models have been formed to assist the user in selecting one of the 16 million colors.

RGB

RGB is directly related to the hardware involved in color displays. Most of us are familiar with this from using the Color Picker package to select colors on the Mac II. The Red, Green, and Blue values are represented as long integers, which are converted into signals driving each of the color guns in the CRT (one for each color). The RGB model is easier for the programmer to implement because it is directly related to the hardware. But it may not be the easiest for the user to use unless he/she understands the RGB model. The other color models are always converted to RGB then the hardware uses a Color LookUp Table (CLUT) to determine which color to display. Also notable about the RGB model, when the values of RGB are equal (R = G = B) the CRT phosphor chromaticities are balanced so that gray is the result. The brightness or darkness of grey is determined by the amplitude of the RGB components as they effect the color guns of the CRT.

HSV

HSV stands for Hue, Saturation and Value. The Hue associates the color with a value in the color spectrum, Saturation represents the vividness of the color with pure colors (such as red, green, and yellow) being called fully saturated colors and grays being called desaturated colors, the Value represents the lightness or darkness of a color. The Brightness field of the Macintosh Color Picker dialog is the equivalent of the HSV value. The Color Picker primarily works using the HSV model. The RGB values are shown also, but only users that understand both color systems will understand how to the HSV and RGB values relate. The RGB system is given so that the user will have an alternate system to choose from just in case the user is familiar with another system other than HSV.

YIQ

The YIQ model has been in use as a television broadcast standard since it was adopted in 1953 by the National Television Standards Committee (NTSC). The purpose of the YIQ model was to provide a means to encode color information in such a way as to be compatible with black and white transmission. Three components of the YIQ model are: Y, an approximation of luminance; I, chrominance information along a blue-green to orange vector; and Q, chrominance information along a yellow-green to magenta vector. A conversion formula was defined by the Commission Internationale de L'Eclairage (CIE):

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1.00 & 0.95 & 0.62 \\ 1.00 & -0.28 & -0.64 \\ 1.00 & -1.11 & 1.73 \end{pmatrix} \cdot \begin{pmatrix} Y \\ I \\ Q \end{pmatrix}$$

LAB

The CIELAB model is an international standard for perceptual uniformity. This one is difficult to explain, and probably just as difficult to understand. This model relates the color differences that a human perceives as being equal Euclidean distances in CIELAB space. This model only approaches its objective because of the varying perception of color that humans have. It is said that CIELAB is better for making fine adjustments in the color selection process.

HSI

HSI processing can be less complicated than other models. Hue values can provide the basic for quantizing and interpreting the color of objects. Tinting, retouch, and special effects can be produced by modifying hue values, or saturation values. Using HSI, all methods for processing and analyzing monochrome images can be applied directly to color images.

HSL, HSV

I don't know the difference between HSI, HSL, HSV models but they have been referred to as separate models in some of the things I've read. HSL is mentioned in *Inside Macintosh*, Vol 5, but not explained. I'll leave the explanation for a future column. I would appreciate any comments in the form of letters that would explain this.

Conversions are available in the Macintosh Color Picker package described in *Inside Macintosh*, Vol. 5 for some of the color models. They are:

```
PROCEDURE CMY2RGB (cColor: CMYColor; VAR rColor: RGBColor);
PROCEDURE RGB2CMY (rColor: RGBColor; VAR cColor: CMYColor);
PROCEDURE HSL2RGB (hColor: HSLColor; VAR rColor: RGBColor);
PROCEDURE RGB2HSL (rColor: RGBColor; VAR hColor: HSLColor);
PROCEDURE CMY2RGB (hColor: HSLColor; VAR rColor: RGBColor);
PROCEDURE CMY2RGB (rColor: RGBColor; VAR hColor: HSLColor);
```

See *Inside Macintosh* for more information on the conversions. It is sufficient here to know that the conversions exist.

Other information on this subject can be found in the following sources (I have not seen all of these sources myself):

1. APPLE COMPUTER, INC., *Inside Macintosh*, Vol 5.
2. SCHWARZ, MICHAEL W., COWAN, WILLIAM B., and BEATTY, JOHN C. *An Experimental Comparison of RGB, YIQ, LAB, HSV, and Opponent Color Models*, ACM Transactions on Graphics, Vol 6, No.2, April 1987, Pages 123-158.
3. BECKER, R. A. and CHAMBERS, J. M. S-*An Interactive Environment for Data Analysis and Graphics*. Wadsworth, 1984.
5. BERK, T., BROWNSTON, L., and KAUFMAN, A. A new color-naming system for graphics languages. *IEEE Comput. Graphil. Appl. 2,3* (May 1982), 37-44
6. BERK, T., BROWNSTON, L., and KAUFMAN, A. A human factors study of color notation systems for computer graphics. *Commun. ACM 25,8* (Aug. 1982), 547-550)
7. BOYNTON, R. M. *Human Color Vision*. Holt, Rinehart and Winston, New York, 1979.
8. CIE. Recommendations on Uniform Color Spaces, Color-Difference Equations, Psychrometric Color Terms. Bureau Central de la CIE (Supplement 2 of CIE Publication 15 (E-1.3.1) 1971), 1978.

9. FOLEY, J. D., and VAN DAM, A. *Fundamentals of Interactive Computer Graphics.* Addison-Wesley, Reading, Mass. 1982.
10. HURVICH, L. M. *Color Vision.* Sinaur, Sunderland, Mass., 1981.
11. WYSZECKI, G., and STILES, W.S. *Color Science: Concepts and Methods, quantitative Data and Formulae,* 2nd ed. Wiley, New York, 1982.
12. JOBLOVE, G. H., and GREENBERG, D. P. Color spaces for computer graphics. *Comput. Graph.* 12,3 (Aug. 1978), 20-25.
13. MUNSELL, A. H. *A Color Notation.* Munsell Color Company, 1939.

```
    REM Hierarchical Menu Demo
    REM ©MacTutor 1988
    REM By Dave Kelly
    WINDOW OFF
    COORDINATE WINDOW
    DEF MOUSE=-1
    everyevent=-1:REM ALL events mask
    False=0:True=NOT False
    DIM dragRect%(3)
    'Find out monitor size just in case we need it
    CALL GETWMGRPORT(WMgrPort&)
    PortTop=PEEK WORD(WMgrPort&+8)
    PortLeft=PEEK WORD(WMgrPort&+10)
    PortBottom=PEEK WORD(WMgrPort&+12)
    PortRight=PEEK WORD(WMgrPort&+14)
    CALL
SETRECT(dragRect%(0),PortLeft+10,PortTop+10,PortRight-
                        10,PortBottom-10)
    REM Setup the EventRecord
    myEvent%=0:REM 'what' is first variable
    message&=0
    when&=0
    where=0:REM y coordinate of point is first
    wherex=0
    modifiers=0
    applemark=&H14
    whichwindow&=0
    charCodeMask&=VAL("&H000000FF")
    keyCodeMask&= VAL("&H0000FF00")
    myWindow&=FN GETNEWWINDOW(29414,0,-1)
    CALL SETPORT(myWindow&)
    TEXT 2,12,0,0
    CLS
    GOSUB "SetUpMenus"
    FLUSHEVENTS
    "EventLoop": REM Main Event Loop
    DO
    CALL SYSTEMTASK
    LONG IF FN GETNEXTEVENT(everyevent,myEvent%)
     SELECT myEvent%
        CASE 0:REM No Event
        CASE 1:REM mousedown
           wResult=FN FINDWINDOW(where,whichwindow&)
           SELECT wResult
              CASE 0:REM inDesk (do nothing)
              CASE 1:REM inMenuBar
                 mResult&=FN MENUSELECT(where)
                 theMenu=FN HIWORD(mResult&)
                 theItem=FN LOWORD(mResult&)
                 SELECT theMenu
                    CASE 255
                       GOSUB "appleID"
                    CASE 134
                       GOSUB "fileID"
                    CASE 2
                       GOSUB "editID"
                    CASE 136
                       GOSUB "StyleID"
                    CASE 137
                       GOSUB "FontID"
                 END SELECT
                 CALL HILITEMENU(0)
              CASE 2:REM inSysWindow
                 CALL SYSTEMCLICK(myEvent%,whichwindow&)
              CASE 3:REM inContent
              CASE 4:REM inDrag
                 CALL DRAGWINDOW(myWindow&, where,
                       dragRect%(0))
              CASE 5:REM inGrow
              CASE 6:REM inGoAway
                 Byte=FN TRACKGOAWAY(myWindow&,where)
                 IF Byte THEN PRINT"
CLOSEWINDOW(myWindow&)"
           END SELECT
        CASE 2:REM mouseup
        CASE 3:REM keydown
           PRINT"key-down"
           charCode&=FN BITAND(message&,charCodeMask&)
           keyCode&=FN BITAND(message&,keyCodeMask&)
           keyCode&=FN BITSHIFT(keyCode&,-8)
           PRINT "Charcode=";charCode&
           PRINT "Keycode=";keyCode&
        CASE 4:REM keyup
        CASE 5:REM autokey
        CASE 6:REM updateEvt
        CASE 7:REM diskEvt
        CASE 8:REM activateEvt
        CASE 10  :REM networkEvt
        CASE 11  :REM driverEvt
        CASE 12  :REM app1Evt
        CASE 13  :REM app2Evt
        CASE 14  :REM app3Evt
        CASE 15  :REM app4Evt
        CASE ELSE
     END SELECT
    END IF
    UNTIL theEnd
    "SetUpMenus"
    APPLE MENU "About Sample..."
    AboutHandle&=FN GETMHANDLE(255)
    FileHandle&=FN GETMENU(134)
    CALL INSERTMENU(FileHandle&,0)
    EDIT MENU 2
    EditHandle&=FN GETMHANDLE(130)
    FormatHandle&=FN GETMENU(135):'Get Format Menu Handle
    StyleHandle&=FN GETMENU(136):'Get Style Menu Handle
    FontHandle&=FN GETMENU(137):'Get Font Menu Handle
    CALL INSERTMENU(FormatHandle&,0)
    CALL INSERTMENU(StyleHandle&,-1):'Style Menu
    CALL INSERTMENU(FontHandle&,-1):'Font Menu
    CALL ADDRESMENU(FontHandle&,CVI("FONT"))
    ' Check default font in Font menu
    fontcnt%=FN COUNTMITEMS(FontHandle&)
    defaultfont$="Geneva"
    FOR i=1 TO fontcnt%
     CALL GETITEM(FontHandle&,i,item$)
     IF item$=defaultfont$ THEN Oldmitem=i:i=fontcnt%+1
    NEXT i
    CALL CHECKITEM(FontHandle&,Oldmitem,True)
    TextStyle=0
    Plain=0
    Bold=1
    Italic=2
    Underline=4
    Outline=8
    Shadow=16
    Condense=32
    Extend=64
    just=0
    CALL DRAWMENUBAR
```

```
    RETURN
    "appleID"
     CALL GETITEM(AboutHandle&,theItem,Var$)
     PRINT Var$
     mrefNum=FN OPENDESKACC(Var$)
     CALL SETPORT(myWindow&)
     RETURN
    "fileID"
     END
    RETURN
    "editID"
     IF NOT FN SYSTEMEDIT(theItem-1) THEN RETURN
     SELECT theItem
        CASE 1:REM undo command
        CASE 3:REM cut command
        CASE 4:REM copy command
        CASE 5:REM paste command
        CASE 6:REM clear command
     END SELECT
    RETURN
    "StyleID"
     CALL GETITEM(StyleHandle&,theItem,Var$)
     CLS: PRINT Var$;" selected"
    RETURN
    "FontID"
     CALL GETITEM(FontHandle&,theItem,Var$)
     CLS: PRINT Var$;" selected"
    RETURN

* ZBasic.R
*Basic.RSRC
???????

Type ZBDK = STR
     ,0
0 Demo by Dave Kelly \0Dver 1 JUN 1988

Type FREF
,128
APPL 0

Type BNDL
,128
ZBDK 0
ICN#
0 128
FREF
0 128

* —— Multifinder events ——
* bit 15 = switcher save screen
* bit 14 = accept suspend resume events
* bit 13 = switcher enable option switch
* bit 12 = can do background on null events
* bit 11 = multifinder aware

Type SIZE = GNRL
     ,-1
.H
4000     ;; $4000 = bits 14 set
.L
128000   ;; (for 150K recomended)
.L
80000 ;; (for 80K minimum)
.I

Type ALRT
  ,1 (0)
80 80 205 400
1
7765

Type ALRT
  ,2 (0)
80 55 205 425
2
7765

Type ALRT
  ,3 (0)
80 55 205 425
3
7765

Type ALRT
  ,4 (0)
80 55 205 425
4
7765

Type ALRT
  ,5 (0)
30 250 82 502
5
7765

Type ALRT
  ,129 (0)
80 80 200 400
129
5555

Type ALRT
  ,130 (0)
80 80 200 400
130
7765

Type ALRT
  ,150 (0)
80 55 200 425
150
7765

Type ALRT
  ,152 (0)
80 80 200 400
152
7765

Type DITL
  ,1 (0)
5

Button
15 245 40 305
Ok

staticText Disabled
55 10 70 290
^0

staticText Disabled
72 10 87 290
^1

staticText Disabled
89 10 104 290
^2

staticText Disabled
106 10 121 290
^3

Type DITL
  ,2 (0)
6
```

```
Button
15 295 40 360
Ok

Button
55 295 80 360
Cancel

staticText Disabled
55 10 70 290
^0

staticText Disabled
72 10 87 290
^1

staticText Disabled
89 10 104 290
^2

staticText Disabled
106 10 121 290
^3

Type DITL
  ,3 (0)
6

Button
15 295 40 360
Yes

Button
55 295 80 360
No

staticText Disabled
55 10 70 290
^0

staticText Disabled
72 10 87 290
^1

staticText Disabled
89 10 104 290
^2

staticText Disabled
106 10 121 290
^3

Type DITL
  ,4 (0)
6

Button
55 295 80 360
Cancel

Button
15 295 40 360
Ok

staticText Disabled
55 10 70 290
^0

staticText Disabled
72 10 87 290
^1

staticText Disabled
89 10 104 290
^2

staticText Disabled
106 10 121 290
^3

Type DITL
  ,5 (0)
4

Button
6 210 45 246
Ok

staticText Disabled
10 65 25 205
^0

staticText Disabled
26 65 41 130
At Line #

staticText Disabled
26 130 41 205
^1

Type DITL
  ,129 (0)
3

Button
15 245 40 305
 OK

staticText Disabled
48 10 64 300
^0

staticText Disabled
66 10 82 320
^1

Type DITL
  ,130 (0)
2

Button
15 245 40 305
 OK

staticText Disabled
48 10 64 300
Not Enough Memory!

Type DITL
  ,150 (0)
5

Button
55 295 80 360
 Stop

Button
15 295 40 360
 Continue

staticText Disabled
48 10 62 290
^0

staticText Disabled
66 10 80 290
^1

staticText Disabled
84 10 98 290
^2

Type DITL
  ,152 (0)
3

Button
15 245 40 305
 Ok

staticText Disabled
48 10 62 300
The Printer Driver Could not be Opened.

staticText Disabled
66 10 80 300
You will not be able to print anything.

* Menus
Type MENU
  ,135 (0)
Format
      Style /\1B!\88
      Font /\1B!\89

Type MENU
  ,134 (0)
      File
      Quit

Type MENU
  ,136 (0)
Styles
      Bold /B
      Italic /I
      Underline /U
      Outline /O
      Shadow /S

Type MENU
  ,137 (0)
Fonts

* A Window template
Type WIND
  ,29414 (0)
Hierarchical Menu Sample
54 78 280 325
Visible NoGoAway
16
0

Type ICN# = GNRL
  ,128 (0)
.H
0001 0000 0002 8000 0004 4000 0009 A000
0013 1000 0026 0800 004C 0400 0098 0200
0133 0100 0267 8080 04CE 4040 09A9 C020
1310 8010 26B8 0008 4C64 3F04 8822 4082
4015 8041 200B 3022 1005 C814 081E 7F8F
0402 3005 0201 0007 0100 8005 0080 6007
0040 1FE5 0020 021F 0010 0407 0008 0800
0004 1000 0002 2000 0001 4000 0000 8000
*
0001 0000 0003 8000 0007 C000 000F E000
001F F000 003F F800 007F FC00 00FF FE00
01FF FF00 03FF FF80 07FF FFC0 0FFF FFE0
1FFF FFF0 3FFF FFF8 7FFF FFFC FFFF FFFE
7FFF FFFF 3FFF FFFE 1FFF FFFC 0FFF FFFF
07FF FFFF 03FF FFFF 01FF FFFF 00FF FFFF
007F FFFF 003F FE1F 001F FC07 000F F800
0007 F000 0003 E000 0001 C000 0000 8000
```

Basic School

True Basic Plays the Simon Game

Dave Kelly
MacTutor Editorial Board
MacTutor Vol. 4 No. 7

The new and improved version of True Basic (ver. 2.01) has just recently started shipping. Some of the new features include: built in support for full color and PICT format images, 68881 coprocessor support, and a more structured environment using modules and workspaces. True Basic also now includes the Runtime package (which used to cost a few hundred dollars), but they are packaging the Macintosh Developer's Toolkit separately. The toolkit contains all of the libraries which are essential for making "real" Macintosh programs (menus, windows, controls etc). It is obvious to any dedicated Macintosh developer that this package should have been included with the True Basic "Basic" package. However, it is true that when True Basic is sold to a school as a training tool, there is not much need for the Macintosh libraries.

This month, I am not going to do an in depth analysis of True Basic 2.01 (Basic Wars, part ??). I'll save that for next month. I would like to show off some of True Basic's new features with respect to Macintosh II color. I'm still impressed by Mac II's color!

True Basic has its own unique set of statements for using color. The SET COLOR n statement (where n is a number from 0 to 255) sets the drawing pen to the color selected. Then a PLOT or some other graphics command may be given to operate in the selected color. Selecting color is not really very unique. ZBasic uses a similar statement (COLOR=n) to select colors.

The Palette Manager is responsible for monitoring and establishing the color environment on the Mac II. The system provides a default palette of colors which applications may use as desired. The default palette is a set of standard colors which may be selected. Color Quickdraw will match the selected color as best as it can, but there may be times when more shades of a particular color are desired than are available in the default palette.

Most video devices including the Mac II video card use an indexed color model. Each pixel value in the video device's memory corresponds to an indexed value in the color table. The color display card uses the RGB value found in the look up table to display the desired color. There are several advantages to using this indexed display method. One is that it is faster. Also, it allows color animation to be used. In color animation, the index value is changed to change display colors; the pixel color itself is not changed. You can especially see the effects of this when using Pixel Paint or Modern Artist and changing the palette using the application. Another way to see the effects of changing the palette is by using the Colorizer software, by Palomar. This is recommended to any Mac II user that wants to make better use of color on the Mac II. *[The Colorizer and the PICT Dective are*

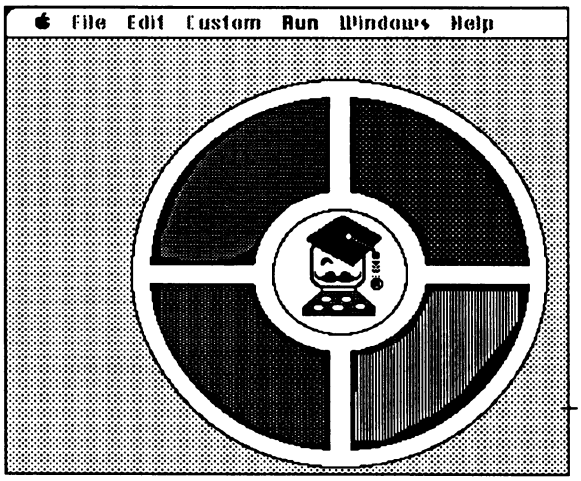


Fig. 1 Our Simon Game in True Basic (This is in color on the Mac II!)

available from MacTutor's Mail Order Store at the back of this issue. -Ed]

Color Palette Manipulation

True Basic allows you to change colors in the color palette. The SET COLOR MIX (x) r,g,b statement sets the color x to the color specified by the red, green, blue values given by r,g and b. In True Basic the values of r,g and b are from 0 to 1 (whereas the hardware is looking for values from 0 to 65535). For the user it is much easier to think in terms of 0 to 1 than 0 to 65535. A SET COLOR MIX (5) .25,0,0 would change color 5 to 25% of the full red value and no green and blue. The number of distinct intensities available on your Mac II depends on the number of colors available. You may use ASK MAX COLOR to determine how many of the colors are available out of the total 256 colors (assuming you can display 256 colors). Actually ASK MAX COLOR returns n-3 colors because some of the colors are already in use by the system. ASK MAX COLOR will only let you know the maximum possible number of colors. SET COLOR MIX is ignored on black and white systems.

Using color is fun as you can see from this months program (see the black and white figure 1; on the Mac II, this Simon game is in color with the background an animated palette). The theme I selected is based on the Simon game (Milton Bradley) which I'm sure you have all seen before. There are a few things which must be done to set up the program if you are starting from scratch. First go to a paint program (color is preferred, but not necessary) and create the main PICT for the game as shown here in figure 2.

© The Definitive MacTutor, Vol. 4

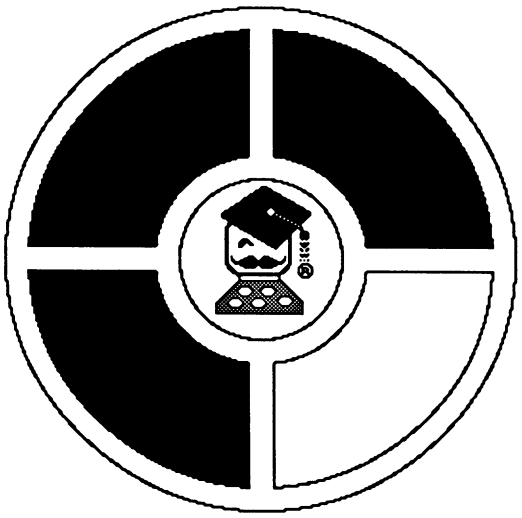


Fig. 2 PICT for the Simon Game

It doesn't matter what colors are used or even the shape of the colored areas. The program will do a fill (FLOOD x,y statement in True Basic) and fill in the correct colors as long as the size of the PICT rectangle remains the same size. If the same color is used as the original, the SET COLOR MIX statement needs to use a different color because the FLOOD statement fills the area till it finds a change in color or it finds the same color as the one being use to fill the area.

Next the PICT should be stored in the clipboard and then run the 'Save Clip' program. The purpose of this program is to convert the clipboard PICT file into a disk PICT file. True Basic could read the PICT file direct, but this way we know that we have only selected the actually PICT and not some of the areas surrounding it. When the 'Save Clip' program asks for a filename, the name of the file as used in the main program should be used (in this case 'Prof Says.PICT' is the filename).

```
! Save Clip
! This program will save the contents of the clipboard to a
file
LIBRARY "Mactools*"
LIBRARY "PictLib*"
DECLARE DEF MacPutFile$
LET doneflag=0
DO until doneflag=1
   CALL Read_clipboard("PICT",s$)
   LET filename$=MacPutFile$(50,50,"Enter Filename to save
as…","","")
   IF filename$<>"" THEN
      CALL Write_pictfile(filename$,s$)
      PRINT "Do another?";
      GET KEY Answer
      IF Answer<>ord("Y") OR Answer<>ord("y") THEN LET
doneflag=1
   ELSE
      LET doneflag=1
   END IF
LOOP
END
```

The main program requires the use of the MacTools and PictLib Libraries. These are provided with the True Basic package. To help True Basic find the libraries (in case they are moved to a strange place) the ALIAS statement should be used. The manual is not too clear on this, but ALIAS statements should be run from a file 'TB Startup' automatically when True Basic is run. This is a satisfactory solution to fix the HFS problems that True Basic 1.0 had.

The next thing that might be different for you is the way that True Basic specifies the window (or screen) coordinates. No matter what kind of computer True Basic is being run on, the coordinates are the same. The coordinates start with 0,0 in the bottom left corner of the screen and go to 1,1 in the top right corner. Then using the SET WINDOW statement any coordinate system may be mapped to the current window. A smaller portion of the screen may be partitioned off to act as a kind of clipping region for text. The PictStuff library seems to only use screen coordinates and doesn't pay much attention to the location of the window. However, by using the CALL Set_Frame statement, the PICT graphics may be displayed anywhere you want. The OPEN #1:screen left, right, bottom, top statement is set up for the Apple RGB Monitor. These values may have to be adjusted if used on other size monitors.

Of course, True Basic statements are used all the way through the program except for the MacTools calls which are used mostly used to display text. A few of the other Macintosh things which are included in the MacTools Library are:

```
MacPenSize(width,height)
MacPenMode(mode)
MacTextFont(font)
MacTextFace(style$)
MacTextMode(mode)
MacTextSize(size)
MacTextBox(left,right,bottom,top,s$,just$)
MacSpaceExtra(extra)
MacGetFontInfo(ascent,descent,widmax,leading)
MacGetFile$(h,v,type$,button$)
MacPutFile$(h,v,pr$,iname$,button$)
MacSysBeep(duration)
```

also other commands for drawing ovals and rectangles.

I am somewhat disappointed that more of the Macintosh Toolbox is not included (built in) to the True Basic system. Of course the Developer Toolkit should include the rest of the Macintosh ROM world, but it would be nice to have it integrated. *[We are still waiting for delivery of the Developer Toolkit. True Basic is sending return postcards to order the library but as yet we have not seen it. -Ed]* If you use any library a lot, you may load it when you startup and have it resident in memory so at least there is a way to simplify the system. I hope to be able to review the Developer Toolkit when it is released. I will have more to say about True Basic next month.

```
! Professor Mac Says…
! By David Kelly
! ©MacTutor, 1988
! With special thanks to Milton Bradley Co.
! For their "Simon" Game

LIBRARY "Mactools*"        ! MacStuff Library
LIBRARY "PictLib*"         ! PICTStuff Library
```

```
DECLARE DEF MacGetFile$
DIM notes(31)
RANDOMIZE
LET skilllevel=8          ! Set up levels
LET level=1
SET BACKGROUND COLOR 10    !  Get a Background color
OPEN #1:screen .25,.67,.165,.75! open a section of screen
WINDOW #1                 ! and use it as a window
SET WINDOW 0,1,0,1
ASK SCREEN a,b,c,d
CALL set_frame(a,b,c,d)

LET filename$="Prof Says.PICT" ! Get the PICT resource
IF filename$<>"" THEN
    CALL Read_pictfile(filename$,s$)
    CALL Draw_string(s$,1)
ELSE
    PRINT "Prof Says.PICT file not found!"
END IF

! Set up the main screen
CALL SetRed
FLOOD .3,.7               ! Fill the shape with Red
CALL SetBlue
FLOOD .7,.7               ! Fill the shape with Blue
CALL SetGreen
FLOOD .3,.3               ! Fill the shape with Green
CALL SetYellow
FLOOD .7,.3               ! Fill the shape with Yellow
! Turn off all shapes
CALL darkred
CALL darkblue
CALL darkgreen
CALL darkyellow
ASK WINDOW a,b,c,d
BOX KEEP a,b,c,d in None$ ! Save PICT so it can be
                  ! restored later
CALL MacTextFont(2)       ! Get New York font
SET TEXT JUSTIFY "center","half"
DO  ! Start the Main program loop
    SET COLOR MIX (0) rnd,rnd,rnd  ! change background color
    CALL SetButtons(level)
    DO
        LET Animationcount=Animationcount+1
        IF Animationcount>=500 then
            SET COLOR MIX (0) rnd,rnd,rnd  ! change the
background color
            LET Animationcount=0
        END IF
        GET MOUSE x,y,s   ! Check for mouse press
        IF s>=1 then
            GET POINT x,y! Get the mouse press
            LET result=0
            CALL PtInRect(x,y,.12,.295,.7,.8,result)! Level
1
            IF result=1 then
                LET skilllevel=8
                LET level=1
                CALL SetStartGameButton(Level)
            END IF
            LET result=0
            CALL PtInRect(x,y,.32,.495,.7,.8,result)! Level
2
            IF result=1 then
                LET skilllevel=14
                LET level=2
                CALL SetStartGameButton(Level)
            END IF
            LET result=0
            CALL PtInRect(x,y,.505,.680,.7,.8,result)
! Level 3
            IF result=1 then
                LET skilllevel=20
                LET level=3
                CALL SetStartGameButton(Level)
            END IF
            LET result=0
            CALL PtInRect(x,y,.705,.880,.7,.8,result)
! Level 4
            IF result=1 then
                LET skilllevel=31
                LET level=4
                CALL SetStartGameButton(Level)
            END IF
            LET result=0
            CALL PtInRect(x,y,.3,.7,.05,.15,result)
            IF result=1 then
                ! Quit Routine
                SET COLOR "White"
                BOX AREA 0,1,0,1
                SET COLOR "Black"
                BOX LINES 0,1,0,1
                SET COLOR "Blue"
                CALL MacTextSize(24)
                PLOT TEXT, AT .5,.6:"Thank you for"
                PLOT TEXT, AT .5,.4:"Reading MacTutor™"
                STOP
            END IF
            LET result=0
            CALL PtInRect(x,y,.3,.7,.2,.3,result)
            IF result=1 then
                ! Play the Game
                BOX SHOW none$ at 0,0
                LET x=0
                LET y=0
                LET numberofnotes=1
                LET tempo=100
                DO until numberofnotes=skilllevel
                    CALL PlaySequence(numberofnotes,tempo,
notes())
                    FOR i=1 to numberofnotes
                    SET COLOR MIX (0) rnd,rnd,rnd
                    GET POINT x,y
                    CALL buttonpress(x,y,selection)
                    IF selection<>notes(i) then
                        CALL MacSysBeep(500)
                        CALL MacSysBeep(500)
                        CALL MacSysBeep(500)
                        EXIT DO
                    END IF
                    NEXT i
                    LET numberofnotes=numberofnotes+1
                    LET tempo=tempo+i*10
                    PAUSE 1
                LOOP
                CALL SetButtons(level)
            END IF
        END IF
    LOOP
LOOP
END

SUB SetButtons(GameLevel) ! Set Game Buttons
    CALL MacTextSize(14)
    SET COLOR "white"
    BOX AREA .3,.7,.05,.15     ! Quit Game Button
    BOX AREA .12,.295,.7,.8    ! Level 1 Button
    BOX AREA .32,.495,.7,.8    ! Level 2 Button
    BOX AREA .505,.680,.7,.8   ! Level 3 Button
    BOX AREA .705,.880,.7,.8   ! Level 4 Button
    SET COLOR "Black"
    BOX LINES .3,.7,.05,.15    ! Quit Game Button
    BOX LINES .12,.295,.7,.8   ! Level 1 Button
    BOX LINES .32,.495,.7,.8   ! Level 2 Button
    BOX LINES .505,.680,.7,.8  ! Level 3 Button
    BOX LINES .705,.880,.7,.8  ! Level 4 Button
```


© The Definitive MacTutor, Vol. 4

```
    SET COLOR "Red"
    CALL SetStartGameButton(GameLevel)
    PLOT TEXT, AT .5,.11:"Quit Game"
    CALL MacTextSize(12)
    PLOT TEXT, AT .195,.77:"1"
    PLOT TEXT, AT .4,.77:"2"
    PLOT TEXT, AT .575,.77:"3"
    PLOT TEXT, AT .825,.77:"4"
END SUB

SUB SetStartGameButton(GameLevel)  ! Set up the Start Game
Button
    SET COLOR MIX (0) rnd,rnd,rnd
    CALL MacTextSize(14)
    SET COLOR "White"
    BOX AREA .1,.9,.2,.3  ! Start Game Button
    SET COLOR "Black"
    BOX LINES .1,.9,.2,.3 ! Start Game Button
    SET COLOR "Red"
    PLOT TEXT, AT .5,.26:"Start Game Level " & STR$(Gamelevel)
END SUB

SUB SetNone           ! Display the PICT with colors off
    BOX SHOW None$ at 0,0
END SUB

SUB SetRed
    SET COLOR MIX (13) 1,0,0
    SET COLOR 13
END SUB
SUB DarkRed
    SET COLOR MIX (13) 0,0,0
    SET COLOR 13
END SUB
SUB SetGreen
    SET COLOR MIX (14) 0,1,0
    SET COLOR 14
END SUB

SUB DarkGreen
    SET COLOR MIX (14) 0,0,0
    SET COLOR 14
END SUB
SUB SetBlue
    SET COLOR MIX (15) 0,0,1
    SET COLOR 15
END SUB
SUB DarkBlue
    SET COLOR MIX (15) 0,0,0
    SET COLOR 15
END SUB
SUB SetYellow
    SET COLOR MIX (16) .9,1,0
    SET COLOR 16
END SUB
SUB DarkYellow
    SET COLOR MIX (16) 0,0,0
    SET COLOR 16
END SUB
SUB FlashRed
    SET COLOR MIX (13) 1,0,0
    LET note$="o5 mf ms c"
    PLAY note$
    PAUSE .5
    SET COLOR MIX (13) 0,0,0
END SUB
SUB FlashGreen
    SET COLOR MIX (14) 0,1,0
    LET note$="o5 mf ms a"
    PLAY note$
    PAUSE .5
    SET COLOR MIX (14) 0,0,0
END SUB

SUB FlashBlue
    SET COLOR MIX (15) 0,0,1
    LET note$="o5 mf ms >a"
    PLAY note$
    PAUSE .5
    SET COLOR MIX (15) 0,0,0
END SUB
SUB FlashYellow
    SET COLOR MIX (16) 1,1,0
    LET note$="o5 mf ms e"
    PLAY note$
    PAUSE .5
    SET COLOR MIX (16) 0,0,0
END SUB
SUB PtInRect(x,y,left,right,bottom,top,Result)
! See if point is in Rectangle
    IF x>left and x<right and y>bottom and y<top then
        LET Result=1
    ELSE
        LET Result=0
    END IF
END SUB

SUB Buttonpress(x,y,selection) ! Handle button press
    CALL PtInRect(x,y,0,.5,.5,1,result)
    IF result=1 then LET selectedcolor=13
    CALL PtInRect(x,y,0,.5,0,.5,result)
    IF result=1 then LET selectedcolor=14
    CALL PtInRect(x,y,.5,1,.5,1,result)
    IF result=1 then LET selectedcolor=15
    CALL PtInRect(x,y,.5,1,0,.5,result)
    IF result=1 then LET selectedcolor=16
    SELECT CASE selectedcolor
    CASE 13
        CALL flashred
        LET selection=1
    CASE 14
        CALL flashgreen
        LET selection=2
    CASE 15
        CALL flashblue
        LET selection=3
    CASE 16
        CALL flashyellow
        LET selection=4
    CASE ELSE
        LET selection=0
    END SELECT
END SUB

SUB PlaySequence(numberofnotes,tempo,notes()) ! Play the notes
    LET tempo$="t"&str$(tempo)
    PLAY tempo$
    FOR i=1 to numberofnotes
        SET COLOR MIX (0) rnd,rnd,rnd
        IF notes(i)=0 then
            LET note=int(4*rnd)+1
        ELSE
            LET note=notes(i)
        END IF
    SELECT CASE note
        CASE 1
            CALL flashred
        CASE 2
            CALL flashgreen
        CASE 3
            CALL flashblue
        CASE 4
            CALL flashyellow
    END SELECT
    LET notes(i)=note
    NEXT i
END SUB
```

Basic School

True Basic & Prototyper Reviewed

True BASIC™

Dave Kelly
Editorial Board
MacTutor Vol. 4 No. 8

True Basic 2.01 Run Time Package

I promised you some more information about the new version of True Basic (2.01). The good news is that you can now create double-clickable applications with the runtime package that is included with the price of True Basic ($99.95 retail).This price includes both a commercial and non-commercial license.There is no longer an extra charge for the license for "unlimited commercial" use. The Bad news is that the Developer's Toolkit is not included with the True Basic interpreter yet. UGH! According to True Basic it will be available soon. The Developer's Toolkit includes all the ROM routines for creating the Macintosh User features we all know and love. As much as I would like the Toolkit to be included with True Basic, they have come a long way. After all, last time I reviewed this the runtime package was priced way out of sight and even the interpreter was overpriced.

You are probably wondering about the runtime package and just what it is. I just received a copy of the runtime package yesterday, but I can pass on to you some information about it. The runtime includes two new libraries of routines, named Bundle and Launch*. Bundle is a "compiled" True Basic program which sets up the signature id of your application. Launch* consists of some routines for getting your application's name and getting the names of files to be opened or printed by the application that were selected in the finder when the application was run. Two application files are included in the runtime package too. RTP is an application shell program which is that base of all applications that you will create. RTP contains the ENTIRE True Basic interpreter and is used as the shell which will 'run' your program after it is attached. The other application is named Binder and is used to combined compiled True Basic programs with the RTP application so that they can now be considered one application. The RTP application (the interpreter) consists of 73K of True Basic standard packages so you should be aware that your programs will always be at least 73K in size. That may be bad news, but at least the application can be created.

True Basic has one of the most extensive sets of library routines available for any language. Some of these routines are very valuable for scientific/engineering applications. True Basic Inc. says that all the original versions of the libraries work with True Basic 2.01 except for the Developer's Toolkit and the Communications Support package. Both of these have been updated to work with version 2.01. The Developer's Toolkit, as I mentioned before, contains all the latest Macintosh ROM routine calls. Some of the libraries available include: Business Graphics Toolkit, Scientific Graphics Toolkit, 3D Graphics, Communications Support, Forms Management, Sorting and Searching, Advanced String Library, and Mathematician's Toolkit. All of these are list price of $69.95 each. They are probably worth it if you think you need the function they perform.

To give you a little feel for the runtime package I have included the following short program which makes use of the launch* library. The program displays the files which were selected in the finder launch of the application.

```
! Runtime Demo

LIBRARY "launch*"
CALL AppParms(name$,message,nfiles)
PRINT "Current Application name is:";name$
SELECT CASE message
CASE 0
     PRINT "Files to be opened:"
CASE 1
     PRINT "Files to be printed:"
END SELECT
FOR i=1 to nfiles
    CALL Appfile(i,filename$,"")
    PRINT filename$
    CALL ClrAppFile(i)
NEXT i
if nfiles=0 then print "NONE"
GET KEY k
END

SAMPLE OUTPUT:

 Current Application name is:TB Runtime Demo
 Files to be printed:
 Hard Disk II:Basic:True Basic 2.0:TB Startup
 Hard Disk II:Basic:True Basic 2.0:Runtime.demo
 Hard Disk II:Basic:True Basic 2.0:Runtime.demo*
```

So just what's so great about True Basic? Actually, the main drawback in the previous version was its inability to be a double-clickable application. As a language, it is excellent, especially in support of scientific applications. On the positive side, True Basic has the strongest version of Basic available on the Mac. The reason I say it is the strongest because the structure and functional capabilities of the language are solid and reliable (I haven't seen any system bombs!) The library routines are very extensive, more extensive than any for MS Basic. Recursion is supported. The 68881 math coprocessor is supported. Color is supported with ability to modify color palettes (July, 1988 MacTutor). Since the runtime package is now made more freely available, you can use True Basic more effectively in real world applications (instead of the class room world that you have without the runtime package).

No Developer's Toolkit Yet

On the negative side, there are a few areas that could have

© The Definitive MacTutor, Vol. 4

been cleaned up a bit. The first negative, and my strongest point is that the Developer's Toolkit is not included with the interpreter. Since I've already mentioned this before I won't dwell on it, but I feel it is a 'must have' if you are doing any serious development. Next there are a few compatibility/ user interface issues some of which you can fix yourself. One problem is the small size of the Standard GetFile/SaveFile Dialog windows. This can be fix with a little ResEdit work. The problem is that the window is only wide enough for short file names (like version 1.0 was when Mac 128K came out). Fortunately this is fixable.

While it is true that True Basic will run with Multifinder, True Basic does not know when to update windows when they have been overwritten by other applications or desk accessories and without the Developer's Toolkit it would be impossible to know that a window needs to be refreshed. True Basic was designed to run in one window which happens to be the size of the screen (any size screen). All graphics commands are automatically scaled to work with whatever size screen (on whatever computer True Basic runs on). Since the main True Basic window is the size of the screen and is not adjustable, it will overlap ALL windows and icons which are running in the background with Multifinder. Since the menus are disabled during True Basic runtime (runtime package), the only way to switch applications is by clicking on the icon in the upper right corner (in the menu bar). The are a few refinements that could be done here to make it more Multifinder compatible.

True Basic for the Macintosh comes with two manuals, the Reference Manual, and the Macintosh™ User's Guide. The Reference Manual is much improved over the old version included with version 1.0. The True Basic statements are listed alphabetically in the back half of the book (part III), but since there are sometimes several statements on a page you have to read carefully to find what you are looking for. The Macintosh™ User's Manual is just what it says it is, a user's manual. The first half of the book tells what each menu of the True Basic application does, then there are some sections telling about a few Macintosh only routines which are included. Included in this is sections on color support, standard GetFile/PutFile stuff, and a few graphics commands for fonts and basic graphics (about like the original MS Basic Toolbox of Graphics commands). The rest of the graphic support (99% of it) is built into the standard True Basic.

Should you buy it? Well, if you do I recommend that you be sure to get the Developer's Toolkit too. I haven't seen the Developer's Toolkit yet (I hope to be able to review it soon!) so I can't form an opinion yet, but since I expect it to be similar or better than the original ToolBox routines (from version 1.0) I would say that it should turn out to be a pretty decent package. True Basic costs $99.95, so with the Toolkit it will be about $170 for the whole enchilada. If you are an educator you may be able to get really good prices on everything (about half price). I would say that there is a lot of potential for more support for True Basic in the future.

ProtoTyper Creates Shell Program Code

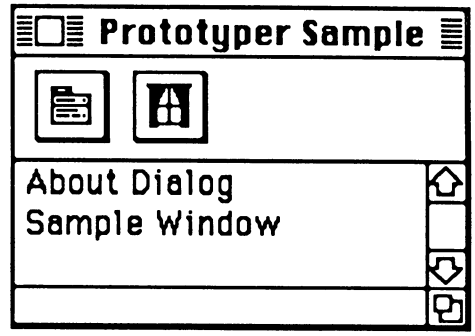
Thank goodness for software developers. At least that's what I say when I don't have the time to write my own software and I need to do something right now. The problem is the length of time it takes to develop software. There are a number of you that, like myself, work full time at a job other than programming and then have family and other responsibilities take you away from that great idea for a software package. Time is valuable, especially when the only time you may have is on weekends and a little time each evening.

One solution to my time schedule problem has been to write most of my programs in Basic because it is "quick and dirty". I can usually see my results during the same weekend or maybe two weekends for a longer program, compared to several weeks working with a full blown Pascal program. That used to be the only solution available to me (except to give it up and not write anything) until recently. It's been different lately.

PRESENTING... PROTOTYPER™

Back at MacWorld Expo in San Francisco I saw Prototyper™ for the first time. Prototyper is one of those truly ingenious ideas whose time has come. If you haven't yet seen it or read about it yet then keep on reading. If you have heard of Prototyper then I don't want it said that it wasn't ever mentioned in MacTutor.

What is Prototyper? Prototyper is a tool which is used to help you to develop the conceptual design of your software and



get you started with your software. It is not all things to all people, but it is a unique tool which may be used to save many hours of development time. Prototyper is not a language compiler; it is a code generation program and resource compiler of a sort. Anyone that uses resources such as menus, windows, dialogs, icons, or controls can make use of Prototyper.

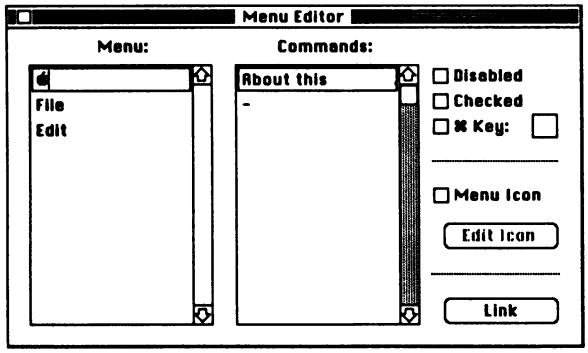
Prototyper is not just another substitute for ResEdit. ResEdit is still very useful for what it does, but Prototyper adds another element of user friendliness to the sometimes hostile world of software development. Let's look at a typical sample Prototyper application. The Prototyper window lists the windows and dialogs that have been defined. Clicking on the menu icon invokes the menu editor. Clicking on the window icon invokes the window/dialog editor. (See fig. 1 previous page)

The Menu editor is similar to other menu editors I have seen before. By clicking in the first column, Menus:, you can select and change the menu to edit. Clicking on an item in the second

column lets you edit each item. The current menu is presented in the menu bar so you can see what you actually get. It's a good thing because the scrolling window is so narrow that longer menu item names can't be seen. A wider window here or a better way to scroll would be valuable, but is not essential to creating a menu successfully. You may check, disable items, or add command-keys. Icons may be added to menus to add a personal touch. Icons may be read in from any application and modified with the icon editor. The Link button links any item to a defined window or dialog. More on that later. Only one menu can be created. (See fig. 2 below)

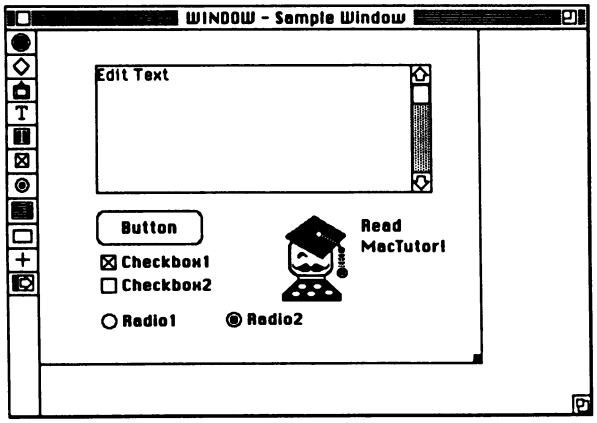
The window editor allows you to create almost any kind of window, dialog, or alert and add text, buttons, scroll bars, active or inactive icons, lists, lines, and rectangles. You should decide before hand whether or not you should use a window, dialog or alert as defined in the guidelines in *Inside Macintosh*. At least once I created a dialog or alert and used it the wrong way and Prototyper bombed! Except for that it was extremely easy to use. (See fig. 3 below)

I created a sample window and sample dialog for this example. The dialog will serve as my About Dialog when the

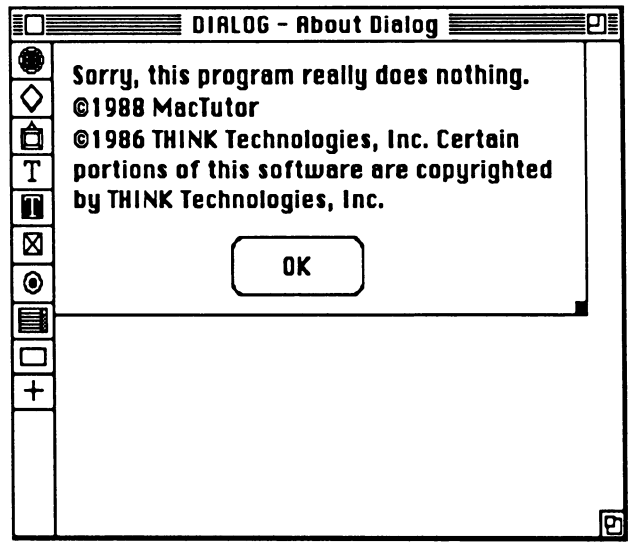


application is created so I have linked the About menu item (in the Menu Editor) to the DIALOG -About Dialog window and set the OK button to close the dialog when pressed. I included some sample buttons and a scroll bar so that you could see how that works. The scroll bars and buttons will move and toggle, but don't effect anything else. They won't do anything until they are updated in the Pascal source code. (See fig. 4 below)

There are a few areas that are deficient that SmethersBarnes



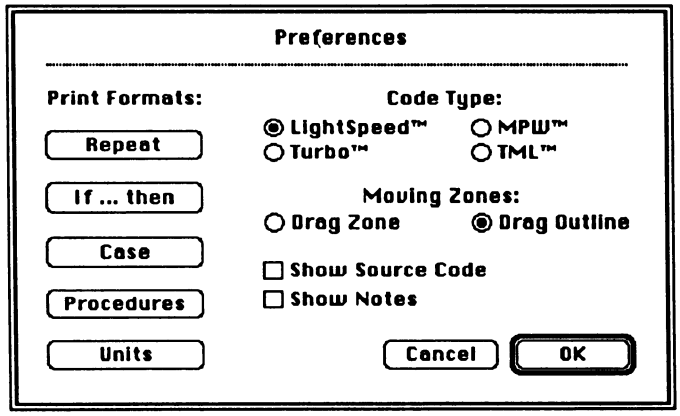
needs to work on to improve the next version. One deficiency is that you can't set up hierarchical menus from within Prototyper. That doesn't mean that you can't do it yourself. It just means that you won't have the menu resources for hierarchical menus set up for you. The next version of Prototyper promises hierarchical menus and support for C. The current version supports four different versions of Pascal, LightSpeed™ Turbo™, MPW™, and TML™. I don't know if that includes the new TML Pascal II which runs under MPW. I'm very pleased with the results of



the LightSpeed code. You set up your preferences and dump the code to the appropriate LightSpeed type files. A resource file is also created along with an RMaker source file (except for PICT resources). (See fig. 5)

Prototyper generates a shell for you which incorporates all of the menus and windows which were created in the Menu and Window Editor. It's too bad that only one menu may be used, but the shell must be able to work with many different application designs. A sample of the LightSpeed Pascal is given on the source code disk for this issue of MacTutor.

I wholeheartedly recommend Prototyper to anyone that would like to write full blown applications, but can't use Basic for one reason or another and doesn't want to take a lot of time writing the shell the application. Some of you might not need



© The Definitive MacTutor, Vol. 4

Prototyper if you already have your own custom application shell, but even then Prototyper is a great way to create the resources. There are things that can and will improve, such as adding hierarchical menus and generation of C source code, but even without some of the enhancements, Prototyper is a major time saver. Improvements will come with time.

```
unit About_Dialog;

{File name:About_Dialog.Pas }
{Function: Handle a dialog }
{History: 6/13/88 Original by Prototyper. }

interface

  procedure D_About_Dialog;

implementation

  const {item numbers for controls in the Dialog}
    I_OK = 1;
    I_Static_Text = 2;
var
    ExitDialog : boolean;  {Flag used to exit the Dialog}
  procedure D_About_Dialog;
    var
      GetSelection : DialogPtr;  {Name of dialog}
      (*ExitDialog : Boolean; declared global for filter
routine *) {Flag to exit dialog}
      tempRect : Rect;           {Temporary tectangle}
      DType : Integer;           {Type of dialog item}
      Index : Integer;           {For looping}
      DItem : Handle;            {Handle to the dialog item}
      CItem, CTempItem : controlhandle;
      sTemp : Str255;       {temp holding}
      itemHit : Integer;         {Get selection}
      temp : Integer; {temp holding}
      Icon_Handle : Handle;     {Temp handle to read Icon }
      NewMouse : Point;     {Mouse loc,tracking Icon presses}
      InIcon : boolean;        {Flag to say pressed in an Icon}
      ThisEditText : TEHandle;  {Handle to Dialogs TE record}
      TheDialogPtr : DialogPeek;  {Pntr to Dialogs def record}

  {This is an update routine for non-controls in the dialog}
  {This is executed after the dialog is uncovered by an alert}
  procedure Refresh_Dialog;  {Refresh dialogs non-controls}
    var
      rTempRect:Rect;     {Temp rectangle used for drawing}
    begin
  end;

    begin        {Start of dialog handler}
      GetSelection := GetNewDialog(2, nil,  Pointer(-1) );
      ShowWindow(GetSelection);          {Open a dialog box}
      SelectWindow(GetSelection);        {Lets see it}
      SetPort(GetSelection);
      TheDialogPtr := DialogPeek(GetSelection);
      ThisEditText := TheDialogPtr^.textH;
      HLock(Handle(ThisEditText));       {Lock it for safety}
      ThisEditText^^.txSize := 12;       {TE Point size}
      TextSize(12);                      {Window Point size}
      ThisEditText^^.txFont := systemFont; {TE Font ID}
      TextFont(systemFont);              {Window Font ID}
      ThisEditText^^.txFont := 0;        {TE Font ID}
      ThisEditText^^.fontAscent := 12;   {Font ascent}
      ThisEditText^^.lineHeight := 12 + 3 + 1;
         {Font ascent + descent + leading}
      HUnLock(Handle(ThisEditText));     {UnLock handle}

      {Setup initial conditions}
      ExitDialog:=FALSE;  {Do not exit dialog handle loop yet}
      repeat          {Start of dialog handle loop}
        ModalDialog(nil, itemHit);  {Wait til item hit}
        GetDItem(GetSelection, itemHit, DType, DItem, tem-
pRect); {Get item information}
        CItem := Pointer(DItem);  {Get the control handle}
        if (ItemHit =I_OK) then  {Handle Button pressed}
          begin
           {?? Code to handle this button goes here}
           ExitDialog:=TRUE;  {Exit dialog when selection made}
          end;                {End for this item selected}
      until ExitDialog;

      {Get results after dialog}
      DisposDialog(GetSelection);    {Flush dialog}
    end;                             {End of procedure}
  end.                              {End of unit}

unit  HandleTheMenus;

{File name :HandleTheMenus.Pas}
{Function: Handle all menu selections.}
{  This procedure is called when a menu item is selected.}
{  There is one CASE statement for all Lists.  There is}
{  another CASE for all the commands in each List.}
{History: 6/13/88 Original by Prototyper.   }

interface

uses
  About_Dialog, Sample_Window, InitTheMenus;

  procedure Handle_My_Menu(var doneFlag:boolean;
theMenu,theItem:integer; var theInput:TEHandle);

implementation

  procedure Handle_My_Menu; {menu selections realtime}
    const
      L_Apple = 201;                    {Menu list}
      C_About_this_Sample = 1;
      L_File = 202;                     {Menu list}
      C_Quit = 1;
      L_Edit = 203;                     {Menu list}
      C_Undo = 1;
      C_Cut = 3;
      C_Copy = 4;
      C_Paste = 5;
      C_Clear = 6;
      C_Select_All = 7;
    var
      DNA:integer;                     {For opening DAs}
      BoolHolder:boolean;              {For SystemEdit result}
      DAName:Str255;                   {For getting DA name}
      SavePort:GrafPtr;    {Save current port when opening DAs}
    begin                    {Start of procedure}
      case theMenu of      {Do selected menu list}
      L_Apple:
        begin
          case theItem of {Handle commands in menu list}
            C_About_this_Sample:
              begin
                D_About_Dialog; {Call dialog for this}
              end;
            otherwise                  {Handle the DAs}
              begin                    {Start of Otherwise}
                GetPort(SavePort);     {Save current port}
                GetItem(AppleMenu, theItem, DAName); {DA}
                DNA := OpenDeskAcc(DAName); {Open DA }
                SetPort(SavePort);     {Restore saved port}
              end;                     {End of Otherwise}
        end;                           {End of item case}
      end;                             {End for this list}

      L_File:
        begin
```

```
        case theItem of
          C_Quit:
            begin
              doneFlag := TRUE;
            end;
          otherwise
            begin
            end;

        end;                              {End of item case}
      end;                                {End for this list}

    L_Edit:
      begin
        BoolHolder := SystemEdit ( theItem - 1 );
              {Do DA editing}
        if not (BoolHolder) then
              {If not a DA then we get it}
          begin     {Handle by using a Case statment}
          case theItem of  {Handle all commands in list}
            C_Undo:
              begin
                {?? ADD IN WHAT THIS COMMAND SHOULD DO}
              end;
            C_Cut:
              begin
                {?? ADD IN WHAT THIS COMMAND SHOULD DO}
              end;
            C_Copy:
              begin
                {?? ADD IN WHAT THIS COMMAND SHOULD DO}
              end;
            C_Paste:
              begin
                {?? ADD IN WHAT THIS COMMAND SHOULD DO}
              end;
            C_Clear:
              begin
                {?? ADD IN WHAT THIS COMMAND SHOULD DO}
              end;
            C_Select_All:
              begin
                {?? ADD IN WHAT THIS COMMAND SHOULD DO}
              end;
            otherwise  {Send to a DA}
              begin    {Start of the Othersize}
              end;     {End of Otherwise}

          end;          {End of item case}
          end;          {End of not BoolHolder}
      end;              {End for this list}

     otherwise
       begin
       end;
    end;                {End for lists}
    HiliteMenu(0);   {Turn menu selection off}
  end;                  {End of procedure Handle_My_Menu}
end.                    {End of unit}

unit  InitTheMenus;

{File name:InitTheMenus.Pas}
{Function: Pull in menu lists from a resource file.}
{    This procedure is called once at program start.}
{    AppleMenu is the handle to the Apple menu, it is also}
{    used in the procedure that handles menu events.}
{ History: 6/13/88 Original by Prototyper.   }

interface

  procedure Init_My_Menus;  {Initialize the menus}

  var
    AppleMenu:MenuHandle;  {Used to handle DAs}

implementation

  procedure Init_My_Menus;    {Initialize the menus}
  const
    Menu1 = 201;               {Menu resource ID}
    Menu2 = 202;               {Menu resource ID}
    Menu3 = 203;               {Menu resource ID}
  var
    tempMenu:MenuHandle;  {Throw away all other menu handles}

  begin                   {Start of Init_My_Menus}
    ClearMenuBar;         {Clear any old menu bars}

    { APPLE menu, used for About & desk accessories.}
    tempMenu := GetMenu(Menu1);   {Get menu from res file}
    AddResMenu(tempMenu,'DRVR');  {Add in DAs}
    InsertMenu (tempMenu,0);      {Insert menu into menu bar}
    AppleMenu:=tempMenu;          {Save menu handle for later}
    { This menu is  File  }
    tempMenu := GetMenu(Menu2);
    InsertMenu (tempMenu,0);
    { This menu is  Edit  }
    tempMenu := GetMenu(Menu3);
    InsertMenu (tempMenu,0);
    DrawMenuBar;        {Draw the menu bar}
  end;                  {End of procedure Init_My_Menus}
end.


Set  RUN OPTIONS to use the resource file
Prototyper_Sample.RSRC
RMaker file to use is Prototyper_Sample.R
The Project should have the following files in it:
     MacPasLib      LSP This is the Pascal library file
     MacTraps       LSP This is for Mac routines to be called}
     ROM85          LSP This is necessary if you use any
Lists}
     ROM85Lib       LSP This is necessary if you use any
Lists}
     ListManager    LSP This is necessary if you use any
Lists}
     About_Dialog        Modal Dialog
     Sample_Window       Window
     InitTheMenus.Pas    This initializes the Menus.
     HandleTheMenus      Handle the menu selections.
     Prototyper_Sample.Pas    Main program

Program  Prototyper_Sample;
{Program name:Prototyper_Sample.Pas   }
{Function:  This is the main module for this program.  }
{History: 6/13/88 Original by Prototyper.   }

uses
  About_Dialog, Sample_Window, InitTheMenus, HandleTheMenus;

  var                             {Main variables}
    myEvent:EventRecord;          {Event record for all events}
    doneFlag:boolean;             {Exit program flag}
    code:integer;                 {Determine event type}
    whichWindow:WindowPtr;        {See which window for event}
    tempRect:Rect;                {Rect for dragging}
    mResult:longint;     {Menu list and item selected values}
    theMenu,theItem:integer;      {Menu list and item selected}
    chCode : integer;             {Key code}
    ch : char;                    {Key pressed in Ascii}
    theInput:TEHandle;            {Used in text edit selections}

  begin                           {Start of main body}

    MoreMasters;      {This reserves space for more handles}
    InitGraf(@thePort);           {Quickdraw Init}
    InitFonts;                    {Font manager init}
```

```
    InitWindows;                    {Window manager init}
    InitMenus;                      {Menu manager init}
    TEInit;                         {Text edit init}
    InitDialogs(nil);               {Dialog manager}
    FlushEvents(everyEvent, 0);     {Clear out all events}
    InitCursor;                     {Make an arrow cursor}
    doneFlag:=FALSE;               {Do not exit program yet}

    Init_My_Menus;                 {Initialize menu bar}
    theInput:=nil; {Init to no text edit selection active}
    Init_Sample_Window;  {Initialize the window routines}
    Open_Sample_Window;  {Open wind routines at program start}

    repeat                          {Start of main event loop}
      if (theInput <> nil) then {See if a TE is active}
        TEIdle(theInput);  {Blink cursor if everything ok}
      SystemTask;       {support of desk accessories}
      if GetNextEvent(everyEvent, myEvent) then
        begin           {Start handling the event}
        code := FindWindow(myEvent.where, whichWindow);
        case myEvent.what of     {Decide type of event}
          MouseDown :           {Mouse button pressed}
            begin               {Handle the pressed button}
              if (code = inMenuBar) then
                begin  {Get menu selection and handle it}
                  mResult := MenuSelect(myEvent.Where);
                  theMenu := HiWord(mResult);
                  theItem := LoWord(mResult);
                  Handle_My_Menu(doneFlag,theMenu,theItem,
theInput);
                end;                {End of inMenuBar}

              if (code = InDrag) then   {window drag area}
                begin       {Do dragging the window}
                tempRect:= screenbits.bounds; {screen l,t,r,b}
                SetRect(tempRect,tempRect.Left+10,
tempRect.Top+25,tempRect.Right-10,tempRect.Bottom-10);
                DragWindow(whichWindow, myEvent.where,
tempRect);
                end;                {End of InDrag}

              if (code = inGrow) then   {In grow area}
                begin           {Handle the growing}
                tempRect:= whichWindow^.portRect;
                      {Get size and position}
                tempRect.Right := tempRect.Right -
tempRect.Left + 1; {Adjust to local}
                tempRect.left := tempRect.Right - 18;
                  {Do for width of right edge}
                tempRect.Bottom := tempRect.Bottom -
tempRect.Top + 1; {Get window height}
                tempRect.top := 0;    {for height of window}
                EraseRect(tempRect);     {Erase old right edge}
                tempRect:= whichWindow^.portRect;
                      {Get window size and position}
                tempRect.Right := tempRect.Right -
tempRect.Left + 1; {Adjust to local}
                tempRect.left := 0;  {Do width of  window}
                tempRect.Bottom := tempRect.Bottom -
tempRect.Top + 1; {Get window height}
                tempRect.top := tempRect.Bottom - 18; {Do for
height of the edge}
                EraseRect(tempRect);  {Erase old bottom edge}
                SetRect(tempRect, 20, 20, 1000, 1000);  {Min
Horz size, Min Vert size, Max Horz size, Max Vert size}
                mResult := GrowWindow(whichWindow,
myEvent.where, tempRect); {Grow it}
                SizeWindow(whichWindow, LoWord(mResult),
HiWord(mResult), TRUE); {Resize to result}
                DrawGrowIcon(whichWindow); {Draw grow Icon
again}
                end;    {End of doing the growing}

              if (code = inGoAway) then {window goaway area}
                begin   {Handle the goaway button}
                If TrackGoAway(whichWindow,myEvent.where) then
                  {See if mouse released in GoAway box}
                  begin     {Handle the GoAway}
                  Close_Sample_Window(whichWindow,theInput);
                      {Close this window}
                  end;        {End of TrackGoAway}
                end;        {End of InGoAway}

              if (code = inContent) then {See if in a window}
                begin     {Handle the hit inside a window}
                if (whichWindow <> FrontWindow) then
                  SelectWindow(whichWindow)
                else
                  begin  {Handle the button in the content}
                  SetPort(whichWindow); {Get ready to draw}
                  Do_Sample_Window ( myEvent,theInput );
                  end;          {End of else}
                end;            {End of inContent}

              if (code = inSysWindow) then {DA?}
                SystemClick(myEvent, whichWindow);

            end;                {End of MouseDown}

          KeyDown,AutoKey:     {Handle key inputs}
            begin         {Get the key and handle it}
              with myevent do     {menu command keys}
                begin
                  chCode := BitAnd(message, CharCodeMask);
                  {Get character}
                  ch := CHR(chCode);     {Change to ASCII}
                  if (Odd(modifiers div CmdKey)) then
                      {See if Command key is down}
                      begin
                      mResult := MenuKey(ch); {menu selection}
                      theMenu := HiWord(mResult);
                      theItem := LoWord(mResult);
                      if (theMenu <> 0) then
                          {See if a list was selected}
                      Handle_My_Menu(doneFlag,theMenu,
theItem,theInput); {Do the menu selection}
                  if ((ch = 'x') or (ch = 'X')) and
(theInput <> nil) then
                      TECut(theInput);  {Handle a TE Cut}
                  if ((ch = 'c') or (ch = 'C')) and (theIn-
put <> nil) then {}
                      TECopy(theInput); {Handle a TE Copy}
                  if ((ch = 'v') or (ch = 'V')) and (theIn-
put <> nil) then
                      TEPaste(theInput); {Handle a Paste}
                  end
              else if (theInput <> nil) then
                  TEKey(ch,theInput);
              end;  {End for with}
            end;        {End for KeyDown,AutoKey}

          UpDateEvt :    {Update event for a window}
            begin       {Handle the update}
              whichWindow := WindowPtr(myEvent.message);
                  {Get the window the update is for}
              BeginUpdate(whichWindow);
                  {Set the clipping to the update area}
              Update_Sample_Window ( whichWindow );
                  {Update this window}
              EndUpdate(whichWindow);
                  {Return to normal clipping area}
            end;          {End of UpDateEvt}

          DiskEvt :      {Disk inserted event}
            begin       {Handle a disk event}
              {Ck for disk event}
              {?? ADD CALL TO HANDLE A DISK INSERTED}
            end;          {End of DiskEvt}
```

```
        ActivateEvt :   {Window activated event}
          begin         {Handle the activation}
            whichWindow := WindowPtr(myevent.message);
                {Get the window to be activated}
            if odd(myEvent.modifiers) then
                {Activate and not DeActivate}
                SelectWindow(whichWindow);
                {Activate the window by selecting it}
          end;          {End of ActivateEvt}
        otherwise
          begin {?? ADDED FOR CATCHING OTHER EVENTS}
          end;          {End of otherwise}
      end;              {End of case}
      end;              {end of GetNextEvent}
   until doneFlag;      {End of the event loop}
end.                    {End of the program}

unit Sample_Window;

{File name: Sample_Window.Pas}
{Function: Handle a Window}
{History: 6/13/88 Original by Prototyper. }

interface

  {Initialize us so all our routines can be activated}
  procedure Init_Sample_Window;

  {Close our window}
  procedure Close_Sample_Window(whichWindow:WindowPtr; var
theInput:TEHandle);

  {Open our window and draw everything}
  procedure Open_Sample_Window;

  {Update our window, someone uncovered a part of us}
  procedure Update_Sample_Window(whichWindow:WindowPtr);

  {Handle action to our window, like controls}
  procedure Do_Sample_Window(myEvent:EventRecord; var
theInput:TEHandle);

implementation

  const
    I_Buttonx1 = 4;                     {Button ID}
    I_Checkbox1 = 6;                    {Checkbox ID}
    I_Checkbox2 = 17;                   {Checkbox ID}
    I_Radio1 = 7;                       {Radio ID}
    I_Radio2 = 18;                      {Radio ID}
    I_Edit_Text = 3;                    {Edit text ID}
    I_Scroll_bar = 5;                   {Scroll bar ID}
  var
    MyWindow:WindowPtr;                 {Window pointer}
    tempRect : Rect;                    {Temporary rectangle}
    Index : Integer;                    {For looping}
    CtrlHandle : controlhandle;         {Control handle}
    sTemp : Str255;             {Get text entered, temp holding}
    R1Control:array[1..2] of ControlHandle; ( Radio button )
    TE_I_Edit_Text: TEHandle; {Text Edit field handle}

{================================}

  {Initialize us so all our routines can be activated}
  procedure Init_Sample_Window;

  begin    {Start of Window initialize routine}
    MyWindow:=nil; {other routines must know not valid yet}
  end;     {End of procedure}

{================================}

  {Close our window}
  procedure Close_Sample_Window;

  begin               {Start of Window close routine}
    if (MyWindow <> nil) and (MyWindow = whichWindow) then
    {Close if this is us}
      begin
      if (theInput = TE_I_Edit_Text) then
         {See if this Text Edit field handle}
      theInput:= nil;      {Clear the handle used}
      DisposeWindow(MyWindow);  {Clear window and controls}
      MyWindow:=nil;  {notify routines window open}
      end;         {End for if (MyWindow<>nil)}
  end;             {End of procedure}

{================================}

  {Update our window, someone uncovered a part of us}
  procedure UpDate_Sample_Window;
  var
    SavePort : WindowPtr;  {Place to save the last port}
    Pic_Handle: PicHandle; {Pict handle for pict loaded }
    sTemp:Str255;     {Temporary string}

  begin          {Start of Window update routine}
    if (MyWindow <> nil) and (MyWindow = whichWindow) then
    {Handle an open when already opened}
      begin
      GetPort(SavePort);     {Save the current port}
      SetPort(MyWindow);     {Set the port to my window}
      Pic_Handle := GetPicture(9 ); {Get Picture into memory}
      SetRect(tempRect, 188,141,247,218);
      if (Pic_Handle <> nil) then
        DrawPicture ( Pic_Handle , tempRect );
         {Draw this picture}

      TextFont(systemFont);  {Set the font to draw in}
      {Draw a string of text, Static Text }
      SetRect(tempRect, 252,146,330,184);
      sTemp :='Read MacTutor!';
      TextBox(Pointer(ord(@sTemp) + 1), length(sTemp),
tempRect, teJustLeft);
      TextFont(applFont); {Set the default application font}

      {Update a TE box, Edit Text }
      SetRect(TempRect,46,27,294,129); {left,top,right,bottom}
      InsetRect(TempRect,-2,-2);        {Surround the TE area}
      FrameRect(TempRect);              {Frame this TE area}
      if (TE_I_Edit_Text <> nil) then
         {Only update if TE area is valid}
        TEUpdate(TempRect,TE_I_Edit_Text); {Update TE area}

      DrawControls(MyWindow); {Draw all the controls}
      DrawGrowIcon(MyWindow);  {Draw the Grow box}
      SetPort(SavePort);                {Restore the old port}
    end;             {End for if (MyWindow<>nil)}
  end;               {End of procedure}

{================================}

  {Open our window and draw everything}
  procedure Open_Sample_Window;
  var
    Index : Integer;                    {For looping}
    dateBounds : Rect;                  {For making lists}
    cSize : Point;                      {For making lists}

  begin              {Start of Window open routine}
    if (MyWindow = nil) then
        {Handle an open when not already opened}
      begin
      MyWindow:= GetNewWindow(1,nil, Pointer(-1));
      SelectWindow(MyWindow);   {Bring window to the front}
      SetPort(MyWindow);    {Prepare to write into our window}
      ( Make a button, Button )
```

```
      CtrlHandle := GetNewControl(I_Buttonx1,MyWindow);
      { Make a checkbox, Checkbox1 }
      CtrlHandle := GetNewControl(I_Checkbox1,MyWindow);
      { Make a checkbox, Checkbox2 }
      CtrlHandle := GetNewControl(I_Checkbox2,MyWindow);
      { Make a radio button, Radio1 }
      R1Control[1] := GetNewControl ( I_Radio1 , MyWindow );
      { Make a radio button, Radio2 }
      R1Control[2] := GetNewControl ( I_Radio2 , MyWindow );
      { Make a scroll bar, Scroll bar }
      CtrlHandle := GetNewControl(I_Scroll_bar,MyWindow);
      {Open a TE box, Edit Text }
      SetRect(TempRect,46,27,294,129); {left,top,right,bottom}
      InsetRect(TempRect,-2,-2);          {Surround the TE area}
      FrameRect(TempRect);                {Frame this TE area}
      InsetRect(TempRect,2,2);      {Restore the original size}
      TE_I_Edit_Text:= TENew(TempRect,TempRect);
         {Create the TE area}
      HLock(Handle(TE_I_Edit_Text));     {Lock the handle}
      TE_I_Edit_Text^^.txFont := geneva; {Font}
      TE_I_Edit_Text^^.fontAscent := 12; {Font ascent}
      TE_I_Edit_Text^^.lineHeight := 12 + 3 + 1;
         {Font ascent + descent + leading}
      HUnLock(Handle(TE_I_Edit_Text));  {UnLock the handle}
      sTemp:= 'Edit Text';
      {Default text string}
      TESetText(Pointer(Ord4(@sTemp)+1),length(sTemp),
TE_I_Edit_Text); {Place default text in the TE area}
      TEActivate(TE_I_Edit_Text); {Make the TE area active}
      TEUpdate(TempRect,TE_I_Edit_Text);
      TextFont(applFont); {Set the default application font}
      UpDate_Sample_Window(MyWindow);   {update draws it}
    end        {End for if (MyWindow<>nil)}
  else
    SelectWindow(MyWindow);   {Already open, so show it}
end;          {End of procedure}

{================================}

  {Handle action to our window, like controls}
  procedure Do_Sample_Window;
  var
    RefCon:integer; {RefCon for controls}
    code:integer;   {Location of event in window or controls}
    theValue:integer;  {Current value of a control}
    whichWindow:WindowPtr;  {Window pntr where event happened}
    myPt:Point;       {Point where event happened}
    theControl:ControlHandle;  {Handle for a control}

  Procedure Do_A_Button;  {Handle a button being pressed}
  begin
    HiliteControl(theControl, 10);       {Darken the button}
    RefCon := GetCRefCon(theControl);   {get control refcon}
    case RefCon of              {Select correct button}
           I_Buttonx1:          {Button, button}
             begin              {start for this button}
               {??? HANDLE THE BUTTON BEING PRESSED HERE}
             end;               {end for this button}

         otherwise      {allow other buttons}
            begin       {start}
            end;        {end of otherwise}
    end;            {end of case}
    HiliteControl(theControl, 0);  {Lighten the button}
  end;   {Handle a button being pressed}

  Procedure Do_A_Checkbox;  {checkbox being pressed}
  var
    Index:integer;     {Index used for radios}

  procedure Clear1RadioGroup;  {clear radios in group 1}
  var
    Index:integer;    {Index used for radios}
  begin               {Start of the clear routine}
    for Index := 1 to 2 do    {Step thru all radios}
      SetCtlValue(R1Control[Index],0);  {Set this to zero}
  end;              {End of the clear routine}

  begin             {Handle a checkbox being pressed}
    RefCon := GetCRefCon(theControl);    {get control refcon}
    theValue := GetCtlValue(theControl); {Get current value}
    theValue := (theValue + 1) mod 2;    {Change value}
    case RefCon of       {Select correct button}
           I_Checkbox1:  {Checkbox1, checkbox}
             begin       {start for this button}
               {??? HANDLE THE CHECKBOX BEING PRESSED HERE}
               SetCtlValue(theControl, theValue);
                 {Set checkbox to new value}
             end;     {end for this checkbox}

           I_Checkbox2:      {Checkbox2, checkbox}
             begin           {start for this button}
               {??? HANDLE THE CHECKBOX BEING PRESSED HERE}
               SetCtlValue(theControl, theValue);
                 {Set checkbox to new value}
             end;            {end for this checkbox}

           I_Radio1:         {Radio1 , radio button}
             begin           {start for this radio button}
               {??? HANDLE THE RADIO BEING SELECTED HERE}
               Clear1RadioGroup;  {Clear all Radio values}
               SetCtlValue(theControl, 1); {Select Radio}
             end;      {end for this radio button}

           I_Radio2:     {Radio2 , radio button}
             begin       {start for this radio button}
               {??? HANDLE THE RADIO BEING SELECTED HERE}
               Clear1RadioGroup;        {Clear all Radio }
               SetCtlValue(theControl, 1); {Select Radio}
             end;        {end for this radio button}
         otherwise {allow other checkboxes and radios}
             begin  {start}
             end;   {end of otherwise}
    end;     {end of case}
  end;       {Handle a checkbox being pressed}

  Procedure Do_A_ScrollBar(code:integer);
        {Handle a ScrollBar being pressed}

    procedure HandleWScrollBar (code, Start, Stop, Increment,
LIncrement : integer; theControl : ControlHandle);
    var
      theValue : integer;   {Value of the scrollbar}
      myPt : Point;     {Returned point from track}

    begin
      theValue := GetCtlValue(theControl); {Get current state}
      if (code = inUpButton) then   {See if in up/left arrow}
        begin
        theValue := theValue - Increment;
        if (theValue < Start) then
          theValue := Start;   {Bump at the start value}
        end;

      if (code = inDownButton) then
            {See if in the down/right arrow}
        begin
        theValue := theValue + Increment;
        if (theValue > Stop) then
          theValue := Stop;    {Bump at the stop value}
        end;

      if (code = inPageUp) then
            {See if in the up/left grey area}
        begin
        theValue := theValue - LIncrement;
            {Subtract the page increment}
        if (theValue < Start) then
```

```
        theValue := Start; {Bump at the Start value}
      end;

    if (code = inPageDown) then {down/right grey area}
      begin
      theValue := theValue + LIncrement; {page increment}
      if (theValue > Stop) then
        theValue := Stop;    {Bump at the Stop value}
      end;

    if (code = inThumb) then    {drag box area}
      begin
      code := TrackControl(theControl, myPt, nil);
           {Let the OS drag it around}
      theValue := GetCtlValue(theControl);
           {Get current state}
      end;
    SetCtlValue(theControl, theValue); {Set new state}
  end;              {End of handle scroll bar}

begin               {Handle a ScrollBar being pressed}
  RefCon := GetCRefCon(theControl);    {get control refcon}
  case RefCon of    {Select correct scrollbar}
          I_Scroll_bar:    {Scroll bar', scroll bar}
            begin         {start for this scroll bar}
              HandleWScrollBar(code,1,10,1,10,theControl);
                {code,Min,Max,Inc,PageInc,handle}
            end;          {end for this scroll bar}

        otherwise
            begin          {start}
            end;           {end of otherwise}
  end;                     {end of case}
end;       {Handle a ScrollBar being pressed}

begin                  {Start of Window handler}
  if (MyWindow <> nil) then
    begin
    code := FindWindow(myEvent.where, whichWindow);
    if (myEvent.what = MouseDown) and (MyWindow =
whichWindow) then
      begin
      myPt := myEvent.where;  {Get mouse position}
      with MyWindow^.portBits.bounds do {Make it relative}
        begin
          myPt.h := myPt.h + left;
          myPt.v := myPt.v + top;
        end;
      SetRect(tempRect,46,27,294,129); {Position of the TE}
      if PtInRect(myPt,tempRect) then {Check for pressed in
the TE  Edit Text }
        begin
        theInput:= TE_I_Edit_Text;
        TEClick(myPt, FALSE, TE_I_Edit_Text);
        end;

      SetRect(tempRect,188,141,247,218); {Position of Pict}
      if PtInRect(myPt,tempRect) then {mouse in pict?}
        begin
          { do somethng for mouse click in pict}
        end;
      end;
    if (MyWindow = whichWindow) and (code = inContent) then
    {for our window}
      begin
      code := FindControl(myPt, whichWindow, theControl);
      if (code <> 0) then    {Check type of control}
        code := TrackControl(theControl,myPt, nil);
      if code = inButton then
        Do_A_Button;                    {Do buttons}
      if code = inCheckBox then
        Do_A_Checkbox;                  {Do checkboxes}
      if (code = inUpButton) or (code = inDownButton) or
(code = inThumb) or  (code = inPageDown) or (code = inPageUp)
then
        Do_A_ScrollBar(code);           {Do scrollbars}
      end;        {End for if (MyWindow=whichWindow)}
    end;          {End for if (MyWindow<>nil)}
  end;            {End of procedure}

{================================}
end.    {End of unit}
```

```
**************************************
*
* RMaker resource file sources.
* File: Prototyper_Sample.R
* History: 6/13/88 Original by Prototyper.
* This file contains the sources for all
* the resources except for Pictures.
*
**************************************
*

Prototyper_Sample.RSRC
APPL????

*
* This is the definition for the DIALOG.
*
Type DLOG
*
     ,2                                  ;;Resource ID
About Dialog                             ;;Dialog title
50  120  189  427                        ;;Top Left Bottom
Right
Visible NoGoAway                         ;;Visible GoAway
1                                        ;;ProcID, dialog def
ID
2                                        ;;Refcon, reference
value
2                                        ;;ID of item list

*
* This is the DIALOG or ALERT item list.
*
Type DITL
*
     ,2          ;;Resource ID
2                ;;Number of controls in list

Button  Enabled  ;;Push button
100  102  129  178    ;;Top Left Bottom Right
OK                 ;;message

StaticText          ;;Static text
9  10  93  295       ;;Top Left Bottom Right
Sorry, this program really does nothing.\0D©1988
MacTutor\0D©1986 THINK Technologies, Inc. Certain portions of
this software are copyrighted by THINK Technologies, Inc.
;;message

*
* This is the definition for the WINDOW.
*
Type WIND
*
     ,1              ;;Resource ID
Sample Window        ;;Window title
41  3  305  350      ;;Top Left Bottom Right
Visible  GoAway      ;;Visible GoAway
0                    ;;ProcID, Window def ID
1                    ;;Refcon, reference value

*
* This is the CONTROL item list.
*
Type CNTL
```


© The Definitive MacTutor, Vol. 4

```
*
   ,4                    ;;Resource ID
Button                   ;;Title for a Button
142  46  168  129       ;;Top Left Bottom Right
Visible                  ;;Initially visible
0                        ;;ProcID (Control definition ID)
4                        ;;RefCon (reference value)
0 1 0                    ;;Min Max Value

   ,6                   ;;Resource ID
Checkbox1                ;;Title for a Checkbox
171  47  193  137       ;;Top Left Bottom Right
Visible                  ;;Initially visible
1                        ;;ProcID (Control definition ID)
6                        ;;RefCon (reference value)
0 1 1                    ;;Min Max Value

   ,17                   ;;Resource ID
Checkbox2                ;;Title for a Checkbox
190  47  212  137       ;;Top Left Bottom Right
Visible                  ;;Initially visible
1                        ;;ProcID (Control definition ID)
17                       ;;RefCon (reference value)
0 1 0                    ;;Min Max Value

   ,7                    ;;Resource ID
Radio1                   ;;Title for a RadioButton
217  47  242  139       ;;Top Left Bottom Right
Visible                  ;;Initially visible
2                        ;;ProcID (Control definition ID)
7                        ;;RefCon (reference value)
0 1 0                   ;;Min Max Value

   ,18                   ;;Resource ID
Radio2                   ;;Title for a RadioButton
216  145  241  237      ;;Top Left Bottom Right
Visible                  ;;Initially visible
2                        ;;ProcID (Control definition ID)
18                       ;;RefCon (reference value)
0 1 1                    ;;Min Max Value

   ,5                    ;;Resource ID
Scroll bar               ;;Title for a Scrollbar
27  293  129  309        ;;Top Left Bottom Right
Visible                  ;;Initially visible
16                       ;;ProcID (Control definition ID)
5                        ;;RefCon (reference value)
1  10  1                 ;;Min Max Value

*
* This is the PICTs for the Window list above.
* Move the pictures from the resource data file
* after this is compiled.
* Type PICT
*

* This Picture is Picture
*  9                          ;;PICT Resource ID

*
* This is the definition for the MENU.
*
Type MENU
*

     ,201                    ;;Resource ID
\14                          ;;APPLE menu title
About this Sample...         ;;item title
(-                           ;;

     ,202                                ;;Resource ID
File                                     ;;menu title
Quit/Q                                   ;;item title

     ,203                                ;;Resource ID
Edit                                     ;;menu title
Undo/Z                                   ;;item title
(-                                       ;;
Cut/X                                    ;;item title
Copy/C                                   ;;item title
Paste/V                                  ;;item title
Clear                                    ;;item title
Select All                               ;;item title
```

Basic School
Matrices in True Basic

True BASIC™

Dave Kelly
MacTutor Editorial Board
MacTutor Vol. 4 No. 9

Using Cramer's Rule and True Basic to Solve Simultaneous Equations

It is only a matter of time until each Engineering or Math student comes across the need to solve simultaneous equations. Fortunately, we live in a time when someone has already figured out the solutions and we can readily use the methods which have been discovered. In this column I hope to show how True Basic's MAT statements can be used with Cramer's Rule to solve simultaneous equations.

In most areas of engineering, the time comes from time to time to solve equations. Many of these solutions can be found using a pencil and paper and a little bit of logical thinking. We learn from algebra that to solve a set of equations, there needs to be one equation generated for each unknown variable. It is the responsibility of the engineer to use his expertise in his field of study to determine what the equations are. The solution of the equations are sometimes trivial, but may take time to manipulate the equations with simple algebra. Eventually the answer is found.

Computers are supposed to make life easier. Right? Well, of course! So let's see how we can solve some equations using True Basic. First, we need a problem to solve: We want to find the current in each of three paths of a circuit. The circuit is shown in figure 1.

To determine the equations which we will use to solve the problem we will use Ohm's law and Kirchhoff's voltage law. For the non-engineer, Ohm's law is a mathematical way to express the relationship between voltage, current, and resistance:

$$E = I * R$$

where R is resistance, E is the voltage across the resistance, I is the current through the resistance. This simple law is basic to a beginning electronics class. Ohm's law is named after a German physicist, George Simon Ohm, who published a pamphlet in 1827 which contained the results of his efforts to describe and relate currents and voltages mathematically. It has since been shown, however, that this result was discovered 46 years earlier in England by Henry Cavendish.

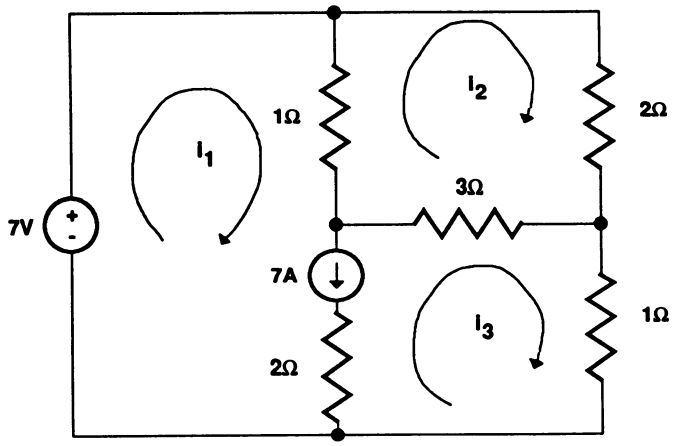


Figure 1

Kirchhoff's voltage law is named after Gustav Robert Kirchhoff, a German university professor who was born about the time Ohm was doing his experimental work. Kirchhoff's law states that **the algebraic sum of the voltages around any closed path in a circuit is zero.** I won't attempt to prove this as there are many electronics books which explain how Kirchhoff's voltage law works.

Our first equation is derived by taking the loop shown in Figure 2. The path of the loop does is not important as long as the path is closed. The equation is:

$$-7 + 1(i_1 - i_2) + 3(i_3 - i_2) + 1i_3 = 0$$

which equates to:

$$i_1 - 4i_2 + 4i_3 = 7.$$

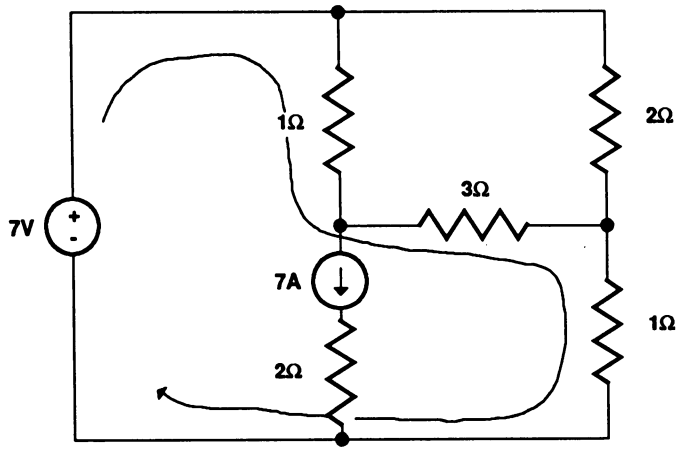


Figure 2

Our second equation is derived by taking the loop where current i_2 is as shown in Figure 1. The equation is:

$$1(i_2 - i_1) + 2i_2 + 3(i_2 - i_3) = 0$$

which equates to:

$$-i_1 - 6i_2 - 3i_3 = 0.$$

The last equation comes from the relationship of the current source and the currents i_1 and i_2:

$$i_1 - i_3 = 7.$$

Now we have three equations and three unknown values to solve for. An easy method to use is given by Cramer's Rule. Gabriel Cramer (1704-1752), was a Swiss mathematician, also known by his book on the theory of curves, which appeared in

© The Definitive MacTutor, Vol. 4

1750 in Geneva. Cramer's Rule states that if we have a system of linear equations

$$a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1,$$
$$a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2,$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots,$$
$$a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n$$

with n equations and n unknowns., the n X n coefficient matrix of the system is denoted by A. The determinant $A \neq 0$, the components of the solution of the system are given by the formula,

$$x_k = \frac{\det B_k}{\det A}, \quad 1 \leq k \leq n,$$

where the matrix B_k is the same as A except that the elements a_{ik}, $1 \leq i \leq n$, in the kth column of A have been replace by the terms b_i, $1 \leq i \leq n$, respectively.

Cramer's Rule says that n simultaneous equations with n unknowns can be determined by taking determinants of the matrix. Each of the unknowns is calculated as follows:

$$\frac{\begin{vmatrix} b_1 & a_{12} & a_{13} & a_{14} & \cdots & a_{1n} \\ b_2 & a_{22} & a_{23} & a_{24} & \cdots & a_{2n} \\ b_3 & a_{32} & a_{33} & a_{34} & \cdots & a_{3n} \\ \cdot & \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & \cdot & & \cdot \\ b_n & a_{n2} & a_{n3} & a_{n4} & \cdots & a_{nn} \end{vmatrix}}{\begin{vmatrix} a_{11} & a_{12} & a_{13} & a_{14} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & a_{24} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & a_{34} & \cdots & a_{3n} \\ \cdot & \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & \cdot & & \cdot \\ a_{n1} & a_{n2} & a_{n3} & a_{n4} & \cdots & a_{nn} \end{vmatrix}} = x_1$$

and

$$\frac{\begin{vmatrix} a_{11} & b_1 & a_{13} & a_{14} & \cdots & a_{1n} \\ a_{21} & b_2 & a_{23} & a_{24} & \cdots & a_{2n} \\ a_{31} & b_3 & a_{33} & a_{34} & \cdots & a_{3n} \\ \cdot & \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & \cdot & & \cdot \\ a_{n1} & b_n & a_{n3} & a_{n4} & \cdots & a_{nn} \end{vmatrix}}{\begin{vmatrix} a_{11} & a_{12} & a_{13} & a_{14} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & a_{24} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & a_{34} & \cdots & a_{3n} \\ \cdot & \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & \cdot & & \cdot \\ a_{n1} & a_{n2} & a_{n3} & a_{n4} & \cdots & a_{nn} \end{vmatrix}} = x_2$$

and

$$\frac{\begin{vmatrix} a_{11} & a_{12} & b_1 & a_{14} & \cdots & a_{1n} \\ a_{21} & a_{22} & b_2 & a_{24} & \cdots & a_{2n} \\ a_{31} & a_{32} & b_3 & a_{34} & \cdots & a_{3n} \\ \cdot & \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & \cdot & & \cdot \\ a_{n1} & a_{n2} & b_n & a_{n4} & \cdots & a_{nn} \end{vmatrix}}{\begin{vmatrix} a_{11} & a_{12} & a_{13} & a_{14} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & a_{24} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & a_{34} & \cdots & a_{3n} \\ \cdot & \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & \cdot & & \cdot \\ a_{n1} & a_{n2} & a_{n3} & a_{n4} & \cdots & a_{nn} \end{vmatrix}} = x_3$$

and

$$\frac{\begin{vmatrix} a_{11} & a_{12} & a_{13} & a_{14} & \cdots & b_1 \\ a_{21} & a_{22} & a_{23} & a_{24} & \cdots & b_2 \\ a_{31} & a_{32} & a_{33} & a_{34} & \cdots & b_3 \\ \cdot & \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & \cdot & & \cdot \\ a_{n1} & a_{n2} & a_{n3} & a_{n4} & \cdots & b_n \end{vmatrix}}{\begin{vmatrix} a_{11} & a_{12} & a_{13} & a_{14} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & a_{24} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & a_{34} & \cdots & a_{3n} \\ \cdot & \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & \cdot & & \cdot \\ a_{n1} & a_{n2} & a_{n3} & a_{n4} & \cdots & a_{nn} \end{vmatrix}} = x_n$$

where

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} & a_{14} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & a_{24} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & a_{34} & \cdots & a_{3n} \\ \cdot & \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & \cdot & & \cdot \\ a_{n1} & a_{n2} & a_{n3} & a_{n4} & \cdots & a_{nn} \end{vmatrix} = \det A$$

The great thing about this is that with True Basic the solution is easy to calculate because of the MAT statements included in True Basic. MAT statements include MAT INPUT (for inputing an array), MAT LINE INPUT, MAT PLOT (plotting to a True Basic chart), MAT PRINT (to print a matrix, MAT READ (read an array from data statements), MAT REDIM (to reset dimensions of a matrix), and MAT WRITE (to write an array to a file).

The MAT statement is an enhancement to Basic that is very welcome in the scientific/engineering community. It has been included for many years in Hewlett Packard Basic (now version 5.0).

Using Cramer's Rule by hand the evaluation of the determinant for i_3 is:

$$i_3 = \frac{\begin{vmatrix} -1 & 6 & 0 \\ 1 & -4 & 7 \\ 1 & 0 & 7 \end{vmatrix}}{\begin{vmatrix} -1 & 6 & -3 \\ 1 & -4 & 4 \\ 1 & 0 & -1 \end{vmatrix}} = \frac{28}{14} = 2$$

evaluation of the other determinants gives i_1= 9 Amps and i_2= 2.5 Amps. The evaluation of these determinants are fairly easy by hand, but with more equations and more unknowns, the it gets to be a lot of number crunching. The program at the end will do the number crunching for an n X n matrix, limited by the amount of memory you have. The number of equations is calculated in a function which uses the factorial function included in the True Basic Libraries to figure how much data is available in the DATA statements. The coefficients are entered into the DATA statements with equation coefficients in the first DATA statement and the solution coefficients in the second DATA statement. For solving for a large number of equations/ unknowns you may break up the DATA into multiple lines.

Once the number of equations is determined, the program calls the sim_solve subroutine to solve the equation. Since the MAT statement does all the work, the subroutine is short and after a couple of MAT assignment statements, the matrix is ready to be crunched. True Basic also supplies us with the det function which will automatically calculate the determinant of an n X n matrix. It sure beats keeping track of a bunch of numbers by hand like I did when I went to school. Calculations are now done and our circuit has been analyzed! Now if they would only put a Mac on my desk...I could use it!

```
! Simultaneous Equation Solver
! Dave Kelly
! ©1988 MacTutor

LIBRARY "Fnmlib"
DECLARE DEF Factrl

DEF Get_Equation_Count              !Read the number of equa-
tions
    WHEN ERROR IN
        LET Count=0
        DO
           READ value
           LET Count=Count+1
        LOOP
    USE
        RESTORE
    END WHEN
    LET n=0
    DO
       LET n=n+1
    LOOP UNTIL Factrl(n)>Count
    LET Count=n-1
    LET Get_Equation_Count=Count
END DEF

! Start Main Program
LET number_of_equations=Get_Equation_Count
DIM a(2,2),b(2,2),c(2,1),x(2)
! Set up all the equations
MAT READ a(number_of_equations,number_of_equations)
MAT READ c(number_of_equations,1)
MAT REDIM x(number_of_equations)
CALL sim_solve(a,b,c,x,number_of_equations)
MAT PRINT x                         ! Print the solutions
DATA 1,-4,4,-1,6,-3,1,0,-1          ! equation coefficients
DATA 7,0,7                          ! equation data
END

SUB sim_solve(a(,),b(,),c(,),x(),number_of_equations)
    IF det(a)=0 THEN EXIT SUB
    FOR j=1 to number_of_equations
        MAT b=a
        FOR i= 1 to number_of_equations
            LET b(i,j)=c(i,1)
        NEXT i
        LET x(j)=det(b)/det(a)
    NEXT j
END SUB
```

Someone told Dave Kelly that he was a programmer. They were kind, but not very accurate. His article *"Hierarchical Menus & Colors Notes"* [VOL. 4 NO. 6] contained more errors than accuracies. He claims that ZBasic source code must be compiled and the resources attached to make use of them. Why not use the function: *"RefNum% =* **FN OPENRESFILE(***FileName$***)"**?

This makes resources available during the programming phase and requires the deletion of only one command before creating the application.

The next glaring error was his slap-dash program that should have shown how to use submenus from ZBasic. *[Excuse me, but I showed A way to to submenus from ZBasic, not THE only way. Actually, I like your way better now that you have shown it to us. Thanks -DK]* As the following program demonstrates, submenus do not require special event handling loops, nor is it necessary to append special resources.

```
'*********************************************
'                SubMenus from ZBasic
'*********************************************
WINDOW OFF
COORDINATE WINDOW
WINDOW 1
TEXT 0,12,0,0
PRINT@(1,1)"MENU#"
PRINT@(1,3)"Item#"
'----------Set The Menu----------
MENU 1,0,1,"File":MENU 1,1,1,"Quit/Q"
'
SubMenu=150       'This is the number we'll use for our
SubMenu
MENU SubMenu,0,1,"I'm outa here"  'We'll delete this from the
menu bar as soon as we've attached it to a menu item.
MENU SubMenu,1,1,"Plain/P;Bold/B<B;Italic/I<I;Outline/
O<O;Shadow/S<S"
```

```
SubMenuHndl&=FN GETMHANDLE(SubMenu)  'Handle to the SubMenu
'
'now delete the name from the menu bar
'CALL DELETEMENU(SubMenu)  '(the menu itself still remains
available)
'
MENU 2,0,1,"Format"        'We'll add the submenu here
'
'Create a menu item with the command key equivalent of
'CHR$(27) and mark the item with the number of the
'SubMenu instead of a "2" for a check mark
'
MenuName$="Style/"+CHR$(27)
MENU 2,1,SubMenu,MenuName$
'
'Use a negative insertion number(-1) and the handle to
'our SubMenu - the "CALL INSERTMENU(SubMenuHandle,-1)"
'should take place immediately after the root menu is created
'
CALL INSERTMENU(SubMenuHndl&,-1)
MENU 2,2,1,"Itz Eazy With Z"  'Add more items and menus if you
like
CALL DRAWMENUBAR  'Redraw the bar to exclude the hidden menu
'----------Events----------
ON MENU GOSUB "Handle Menu"
MENU ON  'We'll just track menu events
"Loop"   'Loop and wait
GOTO "Loop"        'Getting dizzy?
MENU OFF
"Handle Menu"'----------Menu Handling----------
MenuID=MENU(0)
ItemID=MENU(1)
MENU     'Get results of the menu action
IF MenuID=1 AND ItemID=1 THEN END  'End if user selected
"Quit"
PRINT@(9,1)MenuID"       "
PRINT@(9,3)ItemID"       " 'Else-Show menu info
RETURN   'Back to the "Loop"
```

Wake up, Dave. And try to spend a little time developing your programming skills and a little less time throwing rocks.

Regards
Chris Stasny

Well, Chris it seems you have a little chip on your shoulder about something (Do you work for Zedcor?). In my defense, there are many "programmers" out there that have had trouble getting ZBasic event processing to do everything they want it to do. It is true that some of the problems have been fixed as the users have been debugging ZBasic ever since it was released! In my opinion, there is no Basic available today for the Macintosh which is satisfactory for doing serious software development as there are glaring holes in their capabilities. I've had too many phone calls from disgruntled developers trying to find a solution to their problems with no solution in sight. Call it rock throwing if you like, but face it, there isn't any such thing as LightSpeed Basic. Thank you for your letter. We encourage others to share the technology by writing for MacTutor. We are dedicated to the distribution of useful programming information without regard to race, creed or developer status. Chris, we never claim to know everything. We could use more of your good ideas. Write to MacTutor and ask for our authors kit and share some of your "expertise".

Dave Kelly

I'm a new subscriber to MacTutor and have particularly enjoyed your column "Basic School". I'm a BASIC programmer and frankly don't have the time to really learn the other high level languages such as C or Pascal. It is, therefore, heartening to find a source of instruction for BASIC on the MAC.

Do you think it would be feasible to compile your articles into a separate volume? I believe such a book would sell quite well. There are many recreational and small application programmers who both love BASIC and the MAC. Keep up the great work.

Sincerely,
Julian Wan

Thank you for your kind remarks. I've had several people ask for a separate volume of "Basic School". Because of the cost of publishing another compiled book and since we already offer the "Best of MacTutor" Vol 1 and 2 for a reasonable price, "Basic School" will not become a volume of its own. Have heart though, I've used information from C, Pascal, and assembly language columns many times and so having the other language available in "Best of MacTutor" can actually help when writing BASIC programs too. I've also compiled a MacTutor Index HyperCard Stack which can help when you are trying to learn about a specific subject.

Dave Kelly

Basic School

Basic Wars and True Windows

Dave Kelly
Editorial Board
MacTutor Vol. 4 No. 10

Basic Wars Revisited

I promised more information about True Basic. First I would like you to refer back to MacTutor August 1986, February 1987 and March 1987 for the previous episodes of "Basic Wars". The following table gives True Basic (Version 2.0) data for most of the benchmarks used to compare ZBasic and MS Basic 2.0 the last time I ran the benchmark. Please be aware that the math operations were corrected in the March 1987 issue. From a comparison of the benchmarks we can draw a few conclusions. First, True Basic appears to be faster for all math operations even though True Basic is not truly compiled. Only Microsoft Basic (binary) is comparable in math speed, but does not include the accuracy of True Basic. Graphic operations are slower than the MS Basic interpreter. Array look up was comparable to compiled speeds for MS Basic or ZBasic. String handling approaches ZBasic compiled speeds. True Basic ran the Sieve in less time than the MS Interpreter, but slower than any of the compiled Basics.

Still keep in mind that benchmarks are not the only way to determine the strengths of a language. These results show that True Basic performs very well considering that it is not truly compiled to native machine code. It would be much more of a speed competitor if it was compiled. True Basic's strengths are in other areas which cannot be measured only by benchmarks. To some people the other capabilities of True Basic are not enough to justify the slow speed. If the only thing you are looking for in a language is speed then you'll probably be disappointed.

True Basic Benchmark: System 6.0

BENCHMARK	Macintosh Plus	Macintosh II
FOR Loop	1.6 sec.	.388 sec.
Array Lookup	20.67 sec	5.25 sec.
Math Operations		
Addition	22.15 sec.	3.8 sec.
Subtraction	23.23 sec.	3.55 sec.
Multiply	25.55 sec.	3.8 sec.
Divide	39.15 sec.	3.78 sec.
String Operations		
Concatenation	10.3167 sec.	2.35 sec.
Matching	12.15 sec.	3.12 sec.
Graphics		
Line(Basic statement)	61.25 sec.	15.33 sec.
Horizontal line	38.867 sec.	12.3 sec.
Circle	13.4667 sec.	6.63 sec.
Set Pixel (PLOT)	119.3 sec.	37.83 sec.
File I/O (Floppy disk)		
Random READ	24.5 sec.	8.2 sec.
Random WRITE	43.48 sec.	13.7667 sec.
Sequential PRINT	#17.0667 sec.	4.3 sec.
Sequential INPUT	17.45 sec.	4.533 sec.
Sieve (1980 primes)	118.917 sec.	29.0333 sec
Accuracy Benchmark		
Answer:	503.543802149746	503.5438021499906
	1.224999999999966	1.2299999999999960
	.88333 sec.	.166667 sec.

Some of True Basic's strength comes from the structured nature of the ANSI committee standard which it closely follows. Rather than go through a blow by blow account of what each command in True Basic does (you can always read the manual to find out that), I'll tell you what's new since version 1.0 was released. Mostly True Basic is the same very structured, capable language as before. From a scientific, engineering and educational standpoint True Basic offers the best learning environment. It has all the advanced structures of the latest Basics available such as SELECT CASE, DO/LOOP with WHILE and UNTIL, and multi-line IF/ELSE IF/END IF. True Basic is modular in that you can define your own modules which can be loaded as required. Local and global variables are supported. As discussed last month, True Basic has a full set of matrix algebra statements. Arrays are dynamic and resizable. Line numbers are now optional. Recursion is supported. Optional variables may be declared in CALLs.

The built-in editor works very well (unlike the ZBasic editor). It is not the best editor in the world, but is adequate. You have search and replace (Find and Change) and Include (which inserts a file from disk at the cursor point). The annoying little stop light (version 1.0) has been removed. You may select breakpoints in your program where you want to suspend execution to examine variables or do debugging. There are three windows that are used in the interpreter environment. The source window contains the source code, the output window lets you save output from several runs of the program if you want, and the command window lets you type commands specific to managing the interpreter (Actually, True Basic is compiled to b-code for extra speed, but since it is not true native machine code I hesitate to call it a compiler). You can use almost any True Basic statement in direct mode, that is, directly typed into the command window. A simple help system is available, but I feel that unless you are a beginner, the help system is next to useless.

© The Definitive MacTutor, Vol. 4

Script files and DO programs can be set up to configure True Basic or to automate actions. It's sort of like having a a set of exec files that can be executed as short cuts. Since there may be several folders containing library files used, you will want to configure your disk with a TBStartup file containing an ALIAS statement which references all of the places you want True Basic to look for library files. This is a very awkward way to do things, but once it gets set up you just declare your library and True Basic takes care of things for you. (Sort of reminds me of what I don't like about MSDOS). DO programs are used for things such as formatting the source code, getting a cross reference list of variables, or tracing a programs variables. Since you can always write your own DO program you can set it up however you wish.

True Basic supports the 68881 coprocessor which speeds up calculations quite a bit. We've already seen samples of True Basic color capabilities in previous issues of MacTutor.

Macintosh Goodies

Macintosh people want their applications to look right and feel right for the Macintosh user. True Basic provides a library "MacTools*" which contains most of the simple graphics calls:

```
call MacPenSize (width, height)
call MacPenMode (mode)
call MacTextFont (font)
call MacTextFace (style$)
call MacTextMode (mode)
call MacTextSize (size)
call MacTextBox (left, right, bottom, top, s$, just$)
call MacSpaceExtra (extra)
let n = MacStringWidth (s$)
call MacGetFontInfo (ascent, descent,widmax, leading)
call MacInvertRect (left,right,bottom,top)
call MacPaintOval (left,right,bottom,top)
call MacInvertOval (left,right,bottom,top)
call MacFrameRoundRect (left,right,bottom,top,x,y)
call MacPaintRoundRect (left,right,bottom,top,x,y)
call MacInvertRoundRect (left,right,bottom,top,x,y)
let name$=MacGetFile$(h,v,type$,button$)
let name$=MacPutFile$(h,v,pr$,iname$,button$)
call MacSysBeep (duration)
call copy_clipboard
call copy_done
call copy_pictfile (filename$)  supports PICT2 format
call copy_printer(1)
call draw_clipboard(1)
call Read_clipboard(type$,s$)
call Write_clipboard(type$,s$)
call Read_pictfile(filename$,s$)
call Write_pictfile(filename$,s$)
call Draw_string(s$,fit)
```

These are basically the same sort of commands available to MS Basic users as built in statements. Other statements for printer control are also included in "MacTools*" library. True Basic file handling is done using the file names with colons separating the volume names. If the more preferred method of using volume numbers is desired then you should plan on using Mac ROM calls.

The Macintosh Developer's ToolKit supplies the rest of the Macintosh interface. The libraries included in the ToolKit include:

```
TrueWindows*        True Windows
ExecLib*            Semi-portable system routines
HexLib*             Integer arithmetic aids
System*             Low-level interface to System
MacLib*             Trap interfaces for other libraries
DataLib*            Data conversion for ToolKit routines
ColorLib*           Color manager
PickerLib*          Color picker package
ControlLib*         Control manager
DeskLib*            Desk accessory manager
DialogLib*          Dialog manager
DiskLib*            Disk driver
DiskInitLib*        Disk initialization package
EventLib*           Toolbox/System event managers
FileLib*            File manager
FontLib*            Font manager
ListLib*            List manager
MenuLib*            Menu manager
PackageLib*         Package manager
PaletteLib*         Palette manager
QuickLib*           Quickdraw
ResourceLib*        Resource manager
ScrapLib*           Scrap manager
SFPlib*             Standard file package
SoundLib*           Sound manager
SystemLib*          System utilities
TextEditLib*        Text edit
ToolboxLib*         Toolbox utilities
WindowLib*          Window manager
```

As in version 1.0, some calls cannot be mixed with some True Basic commands (especially graphics) very easily. However, this is really not to bad considering the problems we've run into with ZBasic and keeping Macintosh events separate from ZBasic events. Also, the Developer's ToolKit now provides some easy menu and window statements for simple applications. They call this set of routines, TrueWindows.

INTRODUCING "TRUE WINDOWS"

At first it doesn't seem to make a lot of sense to even have the TRUE WINDOWS library available, but in a pinch when you need a quick and dirty application (what Basic is used for the most often), it is good to not have to set up the GetNextEvent Loop. True Windows gives you access to menu, dialog and alert, event handling and window routines. An example using True Windows is included with this column.

True Windows menu statements are set up with DATA statements. The MAT statement makes it easy to read in the entire menu in one statement. The array containing the menu is dimensioned as DIM menu$(0:menus, 0:items) where menus is the number of menus and items is the max number of menu items in any one menu. This set up is fairly simple and easy to use.

A set of 5-6 types of dialog/alert boxes are available. You can also set a timeout for the dialog box so that if there is no user response the program will continue. Using that function is not recommended by the Macintosh user interface guidelines because it allows something to happen which the user has no control over and does not expect to happen.

A True Windows event handler returns menu,refresh, window selection, close box events, window move events, window resize events, up, down, left, right, pageup, pagedown, pageleft, pageright, vscroll, hscroll, single click, double click, triple click and mouse down events as only as often as you ask for them. True Windows or True Basic does not go out and do a GetNextEvent just for the fun of it which in turn messes up your own event handling.

While it is true that True Windows does not do everything, it is a way to get Macintosh looking interfaces without a lot of

extra work. If it doesn't handle the job then the rest of the Developer's Toolkit is sufficient to handle any other jobs that might come up that require the Mac ROM routines. And if Apple comes out with a new call some day it is very simple to go in and add the call yourself just by examining the source code for the Toolkit libraries. The copy of Developer's Toolkit that I have is a prerelease copy, but it appears to be complete in every way.

The hardest thing about using any library is to know the routines that need to be declared in the DECLARE DEF statement so that they can be used in your program. All external functions must be declared. However, the LOAD statement could be used to make the library become a part of the Basic language and thus the DECLARE statement would not be necessary.

Ok, so we know that True Basic can do all this stuff, should I use it for my development project? The answer to this has to be answered yourself, but you should ask your self the following questions:

1. Does my program need to run fast? True Basic speed is satisfactory for many needs, but there are some things that must run faster.

2. Do I care if my application program is greater than 73K? The runtime package uses 73K of disk space. About 35 K of that is in the resource fork of the file and may be shared by more than one application by cutting the resources from the application and pasting into the system file. This is not the preferred method, but could be done if necessary.

3. Do I have a lot of math operations to perform? True Basic is extremely strong in its math support. This is especially true on the Mac II which also includes 68881 math coprocessor support.

4. Do I want full access to the entire Macintosh ROM routines? The Developer's Toolkit does a great job of supporting all of the ROM routines in a way that can be understood. It is easy to add new ROM calls as they are added in the future. You won't have to wait for True Basic's next upgrade to have support of the new features. It would be nice if the ROM support was built into True Basic, but by loading the Library into memory it can become a part of True Basic. Besides, even Lightspeed Pascal requires that Libraries be declared (in the project window) in order to use them.

In my opinion, True Basic has come a long ways by including the runtime package with the Basic system. I would like to see the Developer's Toolkit bundled with the language system, though the price would certainly be near the same price anyway. Currently, the True Basic language system, including Runtime is $99.95. True Basic libraries are $69.95. This makes the combined package cost $169.90. There are only two areas that I wish could be improved upon with True Basic. One would be to have a true compiler that produces real machine code instead of the b-code stuff. I realize that the b-code is designed to be able to transfer applications in True Basic to other computers. As a matter of fact, True Windows routines are supposed to be transferable to other computers such as the Amiga (gasp) which use a window approach. The other area that could improve is in the code size of the runtime package. Some of the size problem could be eliminated if a 'true' compiler were available, that is if it compiled the code efficiently.

```
!Color Picker Demo
!Demonstrates True Basic's True Windows and
!illustrates use of the Color Picker routine
!©1988 MacTutor
!By Dave Kelly
LIBRARY "TrueWindows*","QuickLib*","PickerLib*"
LIBRARY "datalib*", "maclib*", "system*"
DECLARE DEF H,V,GetColor$,UnpackEnvirons$,RGB$
DIM menu$(0:1,0:2)
LET false=0
LET true=1
LET Done=false
CALL tw_init("COLOR")              ! Color isn't supported yet,
but I'm ready for it!
! Find out what kind of computer this is
CALL SysEnvirons(sysEnvRec$,status)
CALL UnpackEnvirons(sysEnvRec$,envversion,machine,sysversion,
processor, hasFPU,hasColorQD,keyboardtype,atversion,
sysvrefnum)
IF hasColorQD<>1 then
   ! we gotta quit, this isn't a Mac II
   LET done=true
   CALL tw_dwarn(3,"Color Quickdraw is not available!  Sorry,
but you can't run this     !","OK",1,result)
END IF
IF done<> true then
   MAT READ menu$
   DATA "",About MacTutor Picker Demo...,""
   DATA File,Get Picker...,Quit
   CALL tw_menu_set(menu$)         ! Set up the menu
   DO
      CALL tw_event(maxnum,type$,window,menunumber,menuitem)
      SELECT CASE type$
      CASE "MENU"
           SELECT CASE menunumber
           CASE 0
                CALL About
           CASE 1
                CALL file(menuitem,done,inrgb$)
           CASE ELSE
           END SELECT
      CASE ELSE
      END SELECT
   LOOP Until Done=true
   CALL tw_menu_clear              ! Get rid of the menus
END IF
CALL tw_cleanup           ! clean up all the TrueWindows stuff
END
SUB about
    ! Set up the About dialog box.
    CALL tw_dwarn(0,"MacTutor Picker Demo|©1988 MacTutor|By
Dave Kelly","OK",1,Result)
END SUB
SUB file(menuitem,done,inrgb$)
    SELECT CASE menuitem
    CASE 1
         ! Set up the upper corner of picker dialog
         CALL SetPt(Point$,0,0)
         ! do picker dialog
         CALL GetColor(point$,"Pick a color, just for
fun!",inrgb$,outrgb$,okflag)
         ! select the color
         IF okflag=1 then LET inrgb$=outrgb$
         CALL unpackRGB(inrgb$,r,g,b)
         LET string$="Color selected
was:|r="&str$(r)&"|g="&str$(g)&"|b="&str$(b)
         ! Display the color in rgb coordinates
         CALL tw_dwarn(1,string$,"OK",1,result)
    CASE 2
         LET done=1
    CASE ELSE
    END SELECT
END SUB
```

Basic School

System Snooping in MS QuickBasic

Dave Kelly
MacTutor Editorial Board
MacTutor Vol. 4 No. 11

MS QuickBasic Arrives!!

It's been a long wait (nearly a year and a half) but Microsoft has finally upgraded their infamous compiler. The best part is that it has really improved!! Microsoft took their interpreter which has always been an excellent product, and combined their compiler to produce what they are calling a new product named QuickBASIC 1.0.

There are some major changes that have taken place and many things have remained the same. The programming environment includes the same "pretty-printing" editor (with some improvements) which was included in earlier versions of the interpreter. There are still two versions of basic, the binary version and decimal version. Programs written using earlier versions of MS BASIC still function properly and in some cases they even work better. When you start listing all of the changes you can easily see why Microsoft chose to call it version 1.0 instead of MS Compiler version 2.0.

The first obvious change is the manual. MS QuickBASIC is now packaged as a shrink wrapped book and disk like their competitors. Most of the fancy packaging on software goes in the trash most of the time anyway. If removing the expense of the packaging lowers the price like it did for this one, I vote that all software companies do the same on all their software. QuickBASIC is priced at a mere $99!! Registered users of either MS Basic Interpreter or Compiler may upgrade to MS QuickBASIC 1.0 for $40. Upgrades are free to any registered user who acquired either of Microsoft's BASIC products for the Macintosh after August 1, 1988. MS QuickBASIC now requires at least 1 MB of memory. The new manual includes two books in one. The user's guide includes a description of how to use the interpreter/compiler functions, a tutorial on developing programs using QuickBASIC, a tutorial on debugging your program, creating a stand-alone application, documentation on how to access the Macintosh toolbox, and documentation for ResEdit (an older version is included with QuickBASIC) and Edit (QuickBASIC includes the latest version of the MDS editor which bombs on the Mac II). The second part of the manual is the BASIC Language Reference which includes a description of Basic variables, syntax, a statement and function reference, and appendix. The appendix contains documentation for invoking ROM routines and the Toolbox library, also some very good information about creating your own Libraries (similar to the old BMLL document) including how to write your own routines from Assembly, C or Pascal. Instructions are also given to install your own icon after compiling your stand-alone application.

The menu bar has now changed. As a matter of fact, when programs are run, a default menu is used which includes File and Edit menus. QuickBASIC removes its own menus so you no longer have to clear the menus when running your program as in earlier versions. Using the compiler options, you may disable the default menu and window for compiled programs.

The 'Search' menu contains new menu items, 'Get Info' and 'Set Info' which allows the user to add new items into the Help data base, or add comments to the help that already exists. The text added is stored in the BASIC source code itself so Help can be different for different programs. It is not known if the Help text can be moved from one source code file to another. The 'Set Info' item cannot be found in the manual and in some respects does not seem to work properly. Once it even corrupted the default help info which is stored in the QuickBASIC application. It is handy to type cmd-I (Get Info) while editing and get a dialog explaining the command that the cursor is pointing to. Another nice feature is the bookmarks. You can set a bookmark to force BASIC to remember the page that you are on so you can return there quickly.

The interpreter is able to help debug your program. For example, before executing the program, breakpoints may be set by inserting a stop sign icon. When the program stops at a breakpoint, or encounters an error, the value of key variables may be displayed. (This smells like Lightspeed doesn't it?) You may then change the value of any variables and continue execution. The highlighted tracing option which was available in earlier versions of MS Basic is still available in QuickBASIC too.

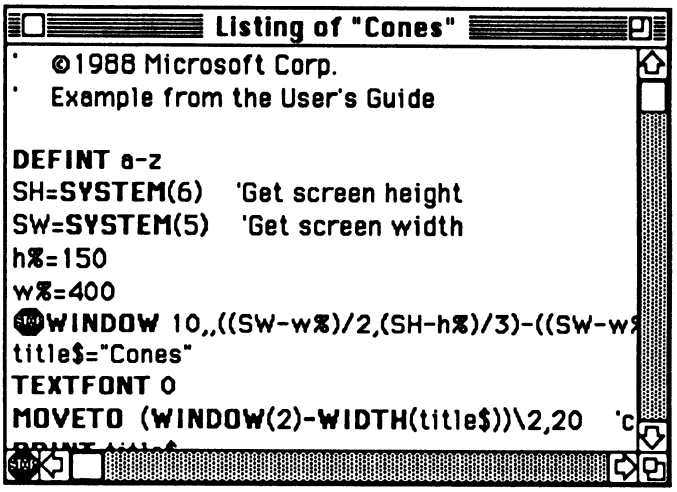


Figure 1. Example Listing

One of the most needed enhancements was HFS support. Now, MS Basic is smart enough to find files anywhere. If you have a LIBRARY statement you don't have to include the full path name because the first time your program is run, if QuickBASIC can't find your library file, it asks you to look for it. Then

it does something even better. After you find the file for QuickBASIC, the path is stored away automatically in your source code file and used the next time the program is run. There may have to be some slight modification to some programs written in earlier versions of BASIC, so QuickBASIC provides a way to disable the File Not Found Dialog option in the 'Options' menu item of the 'Run' menu.

The compiler is built in so it is very easy to get a stand-alone application up and running. There are several options available when compiling, including 68020 and 68881 support if you can use it. Most of the same compiler options are available here that were available in MS Compiler 1.0. As mentioned before, you may choose to use the default window or use the default menus (File and Edit) if needed. Library resources may be included if the 'Include MBPCs and MBLCs' option is selected. (All the CLR Toolbox libraries have been updated and the resource type changed to MBLC (Microsoft Basic Library Code????) resources. The MBPC type is used to denote pure code resources which can be created with some compilers. A utility program is included to scan a QuickBASIC source file and remove library and toolbox resources which are not used by the application. This will reduce the size of the stand-alone application. ALSO, the toolbox library (formerly called ToolLib) is now installed as MBLC resources in the QuickBASIC application. Therefore, all the toolbox statements are available at all times. You now need to specify a type ('&','!', '#') for a library call only after DEFSTR; DEFINT no longer requires a type. For example:

```
DEFINT a-z
DrawText "No more exclamations on library invocations!"
```

There have been several new MBLC routines added to QuickBASIC since the last release of MS BASIC. One of these new statements is the Toolbox statement which allows access to other Toolbox ROM routines which are not specifically available as MBLC routines. You will need Inside Macintosh to make full use of this statement. Routines marked NOT IN ROM are not accessible from QuickBASIC with the ToolBox call. You must write an MBPC or MBLC to do this. The syntax for the ToolBox statement is:

```
ToolBox calltype$, trapnum%, arg1, arg2, ... , argn
```

The calltype$ argument is a BASIC string variable indicating the type of the ROM call. Procedures are indicated by either "P" or "S", Functions returning boolean results use "B", functions returning integers use "W" and functions returning a long integer (LONGINT) use "L". The trapnum% is the trap address of the ROM routine as found in Inside Macintosh. A few samples are given in the manual of some ROM calls. The number of arguments in the ToolBox call will vary depending on the routine that is called.

Sample programs using the ToolBox call are shown below. The first program shows a call to the ROM routine that gets the names of fonts if given an id. The program really isn't of much value except to demonstrate an example use of the ToolBox call. [Make sure to remove all non-printable fonts from the system (i.e. postscript escape fonts) or it will crash; I think it has something to do with zero width fonts.- ed] The second program uses the ToolBox call to get the Macintosh tick count. By using this the benchmark times could have been measured more accurately. The remainder of the program demonstrates the use of the SYSTEM function. The SYSTEM function is used to get system environment information for your program. Getting screen size is one use of the SYSTEM functions available. Other environment information is also available as shown in the program. These functions and the ToolBox calls are explained in more detail in the QuickBASIC manual.

Your old CLR libraries and other libraries you may have created can still be used with QuickBASIC without converting them to the MBLC resource type. Be sure to check to see if the routine already exists in QuickBASIC as a MBLC before using your old routine unnecessarily. You may print lists for comparison purposes with the new version of the Statement Mover program. Statement mover has also been updated to support the MBLC and MBPC resource types.

You're probably wondering how QuickBASIC does in the benchmark tests. Table 1 shows the benchmark results for QuickBASIC , True Basic and ZBasic. ZBasic and True Basic times have appeared in previous issues of MacTutor and were not retested for this table. I have included them here for comparison purposes. Keep in mind that any one benchmark test does not determine how good a language is. All tests were performed on a Macintosh Plus Computer.

BENCHMARK	MS QuickBASIC*	True Basic	ZBasic
FOR Loop (Real)	3 sec	1.6 sec.	15 sec.
Array Lookup (static)	2 sec.	20.67 sec.	1 sec.
Math Operations			
Addition	9 sec.	22.15 sec.	52 sec.
Subtraction	10 sec.	23.23 sec.	64 sec.
Multiply	12 sec.	25.55 sec.	211 sec.
Divide	35 sec.	35 sec.	446 sec.
String Operations			
Concatenation	34 sec.	10.3167 sec.	10 sec.
Pattern Matching	25 sec.	12.15 sec.	6 sec.
Graphics			
Line (Basic stmt)	7 sec.	61.25 sec.	6 sec.
Horizontal line	15 sec.	38.867 sec.	5 sec.
Circle	20 sec.	13.4667 sec.	15 sec.
Set Pixel (PSET or PLOT)	38 sec.	119.3 sec.	19 sec.
File I/O (Floppy Disk)			
Random READ	2 sec.	24.5 sec.	8 sec.
Random WRITE	24 sec.	43.48 sec.	114 sec.
Seql. PRINT#	3 sec.	17.0667 sec.	9 sec.
Seql. INPUT	3 sec.	17.45 sec.	???
Sieve of Eratosthenes	8 sec.	118.917 sec.	7 sec.
Accuracy Benchmark			
x	503.54380215	503.54380215	503.54380215
s	1.23	1.224999999999966	1.23
time	≈1 sec.	.88333 sec.	4 sec.

* NOTE: QuickBASIC compiled binary version used for all

tests except accuracy benchmark.

To obtain optimal performance from compiled QuickBASIC for execution speed it is advisable to use static arrays, turn off array bounds, turn off event checking, generate 68020 and 68881 code if applicable, use SELECT CASE statements instead of IF/THEN/ELSE statements inside loops, and use SHARED (global) arrays when passing to subprograms.

```
' Display Fonts
' ©Copyright 1988 MacTutor
' by Dave Kelly
' This program will display all your system fonts
' with a sample and by name.
' This program uses the ToolBox statement in
' QuickBASIC version 1.0

ToolBox "I"     ' Initialize Toolbox
DEFINT a-z
x=0
FontName$=""
FOR i=0 TO 254     ' Cycle through all the fonts
    TEXTFONT(i)
    GOSUB GetFontsname
    IF FontName$<>"" THEN
        PRINT "This is a sample of font #";
        CALL TEXTFONT(0):PRINT i;"  ";FontName$
        IF x/10 >1 THEN
            x=0:PRINT "Click mouse button to continue..."
            WHILE MOUSE(0)<>1 :WEND:CLS
         ELSE
            x=x+1
        END IF
    END IF
NEXT
WHILE MOUSE(0)<>1:WEND
END
GetFontsname:     'Get the name of the Font with ID = i
TrapNo%=&HA8FF
ToolBox "P",TrapNo%,(i),FontName$
RETURN
```

```
' System Configuration Demo
'©1988 MacTutor
' By Dave Kelly
' This program displays all of the system configuration
' data which is returned by the SYSTEM Function.
' Also the number of tick counts is read from the
' Macintosh directly to get a more accurate time.

ToolBox "I":     'Initialize the toolbox call
start&=0&:finish&=0&
CALL GetTickCount(start&)    :'Call the tick count routine
Screenwidth=SYSTEM(5) :  ' get screenwidth
Screenheight=SYSTEM(6) :  ' get screen height
WINDOW 1,"System Configuration",(4,41)-(Screenwidth-8,Screen-
height-8),1
PRINT "Macintosh System Version:";SYSTEM(0)
machine=SYSTEM(1) :    ' get machine type
SELECT CASE machine
    CASE 0
        machineType$="Unknown"
    CASE 1
        machineType$="Mac 512K"
    CASE 2
        machineType$="Mac Plus"
    CASE 3
        machineType$="Mac SE"
    CASE 4
        machineType$="Mac II"
END SELECT
PRINT "Machine type: ";machineType$
PRINT "Running QuickBASIC version ";SYSTEM(2)
Math=SYSTEM(3):     ' Get Math type
SELECT CASE Math
    CASE 0
        MathType$="Binary Math"
    CASE 1
        MathType$="Decimal Math"
END SELECT
PRINT "Using ";MathType$
environmentFlag=SYSTEM(4)
    IF environmentFlag=0 THEN
        PRINT "Program is being run from the QuickBASIC
environment"
    ELSE
        PRINT "Program has been compiled."
    END IF
CALL GetTickCount(finish&)
PRINT "Total time to run this program was ";(finish&-start&)/
60;" seconds"
END

SUB GetTickCount(Ticks&) STATIC:'Get the real number of ticks
    TrapNo%=&HA975
    Ticks&=0&
    ToolBox "L", TrapNo%,Ticks&
END SUB
```

LETTER BAG...

Your review of True Basic 2.01 was of special interest to me because I've been trying to port a library of thirty astronomical programs from CBasic (remember?) and MS Basic to the Mac. As you pointed out, the math libraries and advanced toolkits make True Basic very useful for scientific tasks. However, the many arbitrary differences in syntax from standard Basics has made the translating task very time-consuming and the source level debugging using DO TRACE is crude compared to MacPascal or LSP.

My greatest disappointment has been the poor speed performance. My Mac has the Turbomax accelerator board and 68881 coprocessor both rated at 16 mhz. Since my work involves heavy use of transcendentals, I use the Savage benchmark; here are some results:

MacPascal 2.0	30.1 seconds	interpreted
True Basic 2.01	29.1 seconds	*
Excel 1.04	12.6 seconds	supports 68881
LS Pascal 1.1	4.7 seconds	compiled

* This is "It's even faster than before." as their ad claims?

Incidentally, they carefully modified the Geneva font so that the capital O and zero [0] are identical; a real 'gotcha' when trying to debug. —Leonard Kalish

Maybe QuickBASIC will be better for you as it can be compiled. Also if you use CLR Libraries MathStatLib you get some of the math routines back. The difference in the Basic syntax is because of the ANSI standard influence of Thomas E. Kurtz, one of True Basic's authors. It is interesting that most of the other Basics out there are enough different from the ANSI standard to make translating a difficult task. Thanks for your comments. DK.

ßasic School

QuickBasic's Statement Mover

Dave Kelly
Editorial Board
MacTutor Vol. 4 No. 12

Statement Mover Fix

During the past month, I've been looking at the demo programs included with MS QuickBASIC 1.0. The programs exercise most of (but certainly not all) the syntax of QuickBASIC. Also utility programs are provided which can be used to help your development are:

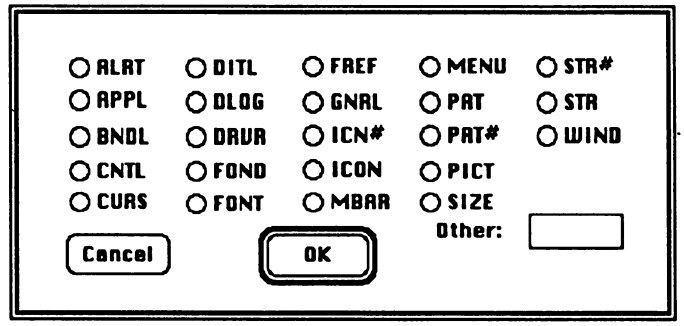
'AlertMover'	For copying Alert/Dialog resources
'Cursor Editor'	For defining your own cursors
'Demo Resources'	containing Alert resources
'extra MBLC Remover'	removes unnecessary library resources
	from your application programs.
'Pattern Editor'	For defining your own patterns
'Print Listing File'	Print listings created at compile time.
'renumber CODEs'	avoids code id conflicts
'SortSub'	moves sub programs to the end of a
Basic	source code file
'Statement Mover'	moves resources especially Library
resources.	

Missing from this list is a variable cross reference program and a program to compare the differences between two text files. Comparing differences between files is somewhat complex when programs don't have line numbers to compare with.

The new version of 'Statement Mover' is much improved over the version included with MS BASIC 3.0 and the CLR Libraries. You can now copy most any kind of resource that has a resource name attached. But, don't open up an open resource file or you get a bomb! That means you can't open the System File or the 'Statement Mover' application. I haven't worked on a fix for that one yet, but I have made a minor improvement to 'Statement Mover' which uses a dialog resource.

The routine I have written replaces the NewTypeDialog subprogram in 'Statement Mover'. First the program's volume reference number is retrieved and the dialog is displayed. Then a loop is set up to retrieve changes in the modal dialog. When a radio button is pressed the routine responds to it and updates the other buttons. The buttons represent a set of resource types which might be moved from one program or file to another. The edit field is updated with the resource type so that it will be passed when the routine is exited.

The dialog resource may be created with ResEdit or any other program that allows you to create dialogs. I used ProtoTyper™ to create the dialog so that I could try it out before I copied it into my QuickBASIC program. Because I had so many buttons in my dialog I used ResEdit to adjust the order of the buttons so that the 'OK' button would be the first one (ProtoTyper™ sets them up in the order that they were drawn and you can't change the order without starting over).



Another thing that this dialog resource shows is a way to create the bold default buttons. Normally in Pascal or C you can draw the bold with the FrameRoundRect statement after defining the dialog item as a user item. A method that works is to define a PICT resource that looks like the bold roundrect and position it around the button which will be the default (usually the 'OK' button). I believe that you can do it this way from other languages too, although Inside Macintosh vol. 1 says to use the user items.

```
' NewTypeDialog Subprogram
' Replaces NewTypeDialog in Statement Mover Program
' Subprogram by Dave Kelly
' Portions derived from Prototyper.

'Use these lines to test the program
' cancel%=0
' NewType$=""
' CALL NewTypeDialog(NewType$,cancel%)
' PRINT NewType$
' END

SUB NewTypeDialog(NewType$,cancel%) STATIC
    False=0:True=NOT False
    ExitDialog =False  ' Flag used TO EXIT the DIALOG

    Selection&=0  : '  DialogPtr;NAME of DIALOG
    index%=0 :'  FOR looping
    EditString$="" :' GET Text entered, temp holding
    itemHit%=0 : ' GET selection

    ' Start of DIALOG handler
    vref%=SYSTEM(7): '  Get program's volume reference number.
    NewType$=""
    GetNewDialog vref%,10000,Selection&    ' Bring in the
DIALOG resource

    ExitDialog=False  '   Do NOT EXIT DIALOG Handle loop yet
    itemHit%=0
    WHILE ExitDialog=False ' Start of DIALOG Handle loop
        ModalDialog Selection&, itemHit%   :'  WAIT until an
item IS hit
        SELECT CASE itemHit%
            CASE 1        ' Handle the OK BUTTON being pressed
                ExitDialog=True '   (EXIT the DIALOG when This
```


© The Definitive MacTutor, Vol. 4

```
selection IS made)
                cancel%=False

            CASE 2   '      Handle the Cancel BUTTON being
pressed
                ExitDialog=True  '  (EXIT the DIALOG when This
selection IS made)
                SetDialogText Selection&,4,""
                cancel%=True

            CASE 5 TO 27
                FOR index%=5  TO 27 ' CLEAR ALL other radios
                    IF index%<>itemHit% THEN SetDialogBut
Selection&,index%,1
                NEXT index%
                ' Setup the Dialog selections
                SetDialogBut Selection&,itemHit%,2
                IF itemHit%=5 THEN EditString$="ALRT"
                IF itemHit%=6 THEN EditString$="APPL"
                IF itemHit%=7 THEN EditString$="BNDL"
                IF itemHit%=8 THEN EditString$="CNTL"
                IF itemHit%=9 THEN EditString$="CURS"
                IF itemHit%=10 THEN EditString$="DITL"
                IF itemHit%=11 THEN EditString$="DLOG"
                IF itemHit%=12 THEN EditString$="DRVR"
                IF itemHit%=13 THEN EditString$="FOND"
                IF itemHit%=14 THEN EditString$="FONT"
                IF itemHit%=15 THEN EditString$="FREF"
                IF itemHit%=16 THEN EditString$="GNRL"
                IF itemHit%=17 THEN EditString$="ICN#"
                IF itemHit%=18 THEN EditString$="ICON"
                IF itemHit%=19 THEN EditString$="MBAR"
                IF itemHit%=20 THEN EditString$="MENU"
                IF itemHit%=21 THEN EditString$="PAT "
                IF itemHit%=22 THEN EditString$="PAT#"
                IF itemHit%=23 THEN EditString$="PICT"
                IF itemHit%=24 THEN EditString$="SIZE"
                IF itemHit%=25 THEN EditString$="STR#"
                IF itemHit%=26 THEN EditString$="STR "
                IF itemHit%=27 THEN EditString$="WIND"
                SetDialogText Selection&,4,EditString$

            CASE ELSE
                FOR index%=5  TO 27 ' CLEAR ALL radio buttons
                    SetDialogBut Selection&,index%,1
                NEXT index%

        END SELECT
    WEND   ' Handle DIALOG items until EXIT selected
    GetDialogText Selection&,4,NewType$
    DisposeDialog Selection&  '       Flush the DIALOG out of
memory
END SUB '     END of unit
```

The following Dialog resource file should either be created with Compiled with a resource compiler or recreated with ResEdit or Prototyper.

```
resource  'PICT' (600, purgeable)
{
    49,
    (-1, -1, 35, 70),
    (17; 1; 160; 0; 130; 1; 0; 10; 255; 255; 255; 255; 0
    ; 35; 0; 70; 7; 0; 3; 0; 3; 11; 0; 21; 0
    ; 21; 64; 0; 1; 0; 1; 0; 35; 0; 70; 160; 0
    ; 131; 255)
};

resource  'DITL' (10000 , "NewTypeDialog")
{
  {
    (130, 146, 156, 207),
      Button (enabled, "OK");
    (130, 26, 155, 88),
      Button (enabled, "Cancel");
    (120, 246, 136, 297),
      StaticText (enabled, "Other:");
    (120, 306, 136, 357),
      EditText (enabled, "");
    (20, 26, 42, 90),
      RadioButton (enabled, "ALRT");
    (40, 26, 60, 81),
      RadioButton (enabled, "APPL");
    (60, 26, 82, 90),
      RadioButton (enabled, "BNDL");
    (80, 26, 102, 90),
      RadioButton (enabled, "CNTL");
    (100, 26, 120, 81),
      RadioButton (enabled, "CURS");
    (20, 96, 42, 160),
      RadioButton (enabled, "DITL");
    (40, 96, 62, 160),
      RadioButton (enabled, "DLOG");
    (60, 96, 82, 160),
      RadioButton (enabled, "DRVR");
    (80, 96, 102, 160),
      RadioButton (enabled, "FOND");
    (100, 96, 122, 160),
      RadioButton (enabled, "FONT");
    (20, 166, 40, 221),
      RadioButton (enabled, "FREF");
    (40, 166, 62, 230),
      RadioButton (enabled, "GNRL");
    (60, 166, 80, 221),
      RadioButton (enabled, "ICN#");
    (80, 166, 100, 221),
      RadioButton (enabled, "ICON");
    (100, 166, 120, 221),
      RadioButton (enabled, "MBAR");
    (20, 236, 40, 296),
      RadioButton (enabled, "MENU");
    (40, 236, 62, 300),
      RadioButton (enabled, "PAT ");
    (60, 236, 80, 291),
      RadioButton (enabled, "PAT#");
    (80, 236, 102, 300),
      RadioButton (enabled, "PICT");
    (100, 236, 120, 291),
      RadioButton (enabled, "SIZE");
    (20, 306, 40, 361),
      RadioButton (enabled, "STR#");
    (40, 306, 62, 370),
      RadioButton (enabled, "STR ");
    (60, 306, 80, 361),
      RadioButton (enabled, "WIND");
    (123, 139, 161, 212),
      Picture (enabled, 600)
  }
};

resource  'DLOG' (10000 , "NewTypeDialog")
{
    (50, 120, 223, 504),
    1,
    invisible,
    noGoAway,
    0x1,
    10000,
    "NewTypeDialog"
};
```

I came across a bug in the AlertMover program which I include a fix here. The problem was that AlertMover would not save to an existing file. There are some fundamental problems

with some of the sample programs on the QuickBASIC examples disk which are related to finding the filename of the application program. Microsoft added the SYSTEM(7) function after the manual was printed and so was probably not known or available at the time the demo programs were written. Changes could be made to the other programs in a similar manner. Here is the fix:

WAS:

```
'    AlertMover
'    © Sharon Zardetto Aker
'    © 1988 by Microsoft Corp.
'    Modified from Sharon Zardetto Aker's original to take
'    advantage of new QB features.
'
'_____________________________
'
   CLEAR,100000&
   OPTION BASE 1

SH=SYSTEM(6)    'Get screen height
SW=SYSTEM(5)    'Get screen width
```

IS:

```
'    AlertMover
'    © Sharon Zardetto Aker
'    © 1988 by Microsoft Corp.
'    Modified from Sharon Zardetto Aker's original to take
'    advantage of new QB features.
'    Modified again so that it would create new files properly
'    by Dave Kelly 9/17/88
'_____________________________
'
   CLEAR,100000&
   OPTION BASE 1

DIM SHARED IOPB%(60),fil$    :'used by new subprogram
***************

SH=SYSTEM(6)    'Get screen height
SW=SYSTEM(5)    'Get screen width
```

WAS:

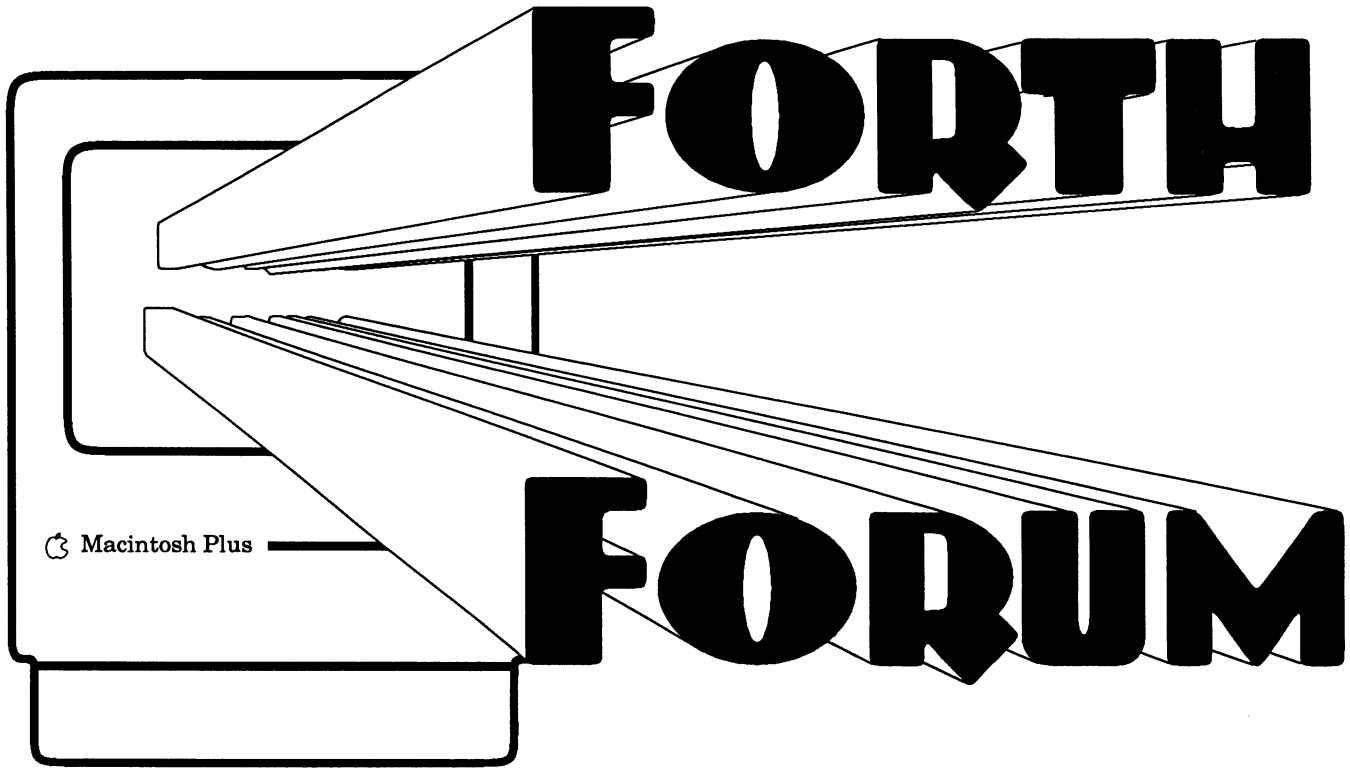
```
  IF SYSTEM(4) THEN    'If compiled...
       openResFile "Demo Resources",ref%
    ELSE    'if interpreted, get File reference number of this
source
       ref%=PEEKW(&HA5A)
    END IF
    ON BREAK GOSUB quit :BREAK ON
    MENU 1,0,1,"File"
```

IS:

```
  IF SYSTEM(4) THEN    'If compiled...
       PgmPath$=""
       ref%= SYSTEM(7)
       GetPathName ref%, PgmName$
       parse PgmName$,path$    'separate file name from the
path
       OpenResFile PgmName$,ref%
    ELSE    'if interpreted, get File reference number of this
source
       ref%=PEEKW(&HA5A)
    END IF
    ON BREAK GOSUB quit :BREAK ON
    MENU 1,0,1,"File"
```

Then add the following subprograms to the end: (These routines were taken from the 'GetPathNames' demo program which is included with MS QuickBASIC version 1.0). This is a great way to find the pathnames of files you are using.

```
'_____________________________
'   Gets the full path name of a file given the file reference
'  number of file.  The file reference may be obtained from
'    OpenResfile, SYSTEM(7), GetChanRefNum, or various low
'    memory locations.
'
SUB GetPathName(ref%,path$) STATIC
    fil$="" : Folder$=""
    GetFCBInfo IOPB%(0),fil$,ref%
    path$=fil$
    DirID&=IOPB%(29)*65536&+IOPB%(30)
    VRefNum%=IOPB%(26)
    WHILE DirID&<>1
        GetCatInfo IOPB%(0),Folder$,DirID&,VRefNum%
        path$=Folder$+":"+path$
        DirID&=IOPB%(50)*65536&+IOPB%(51)
    WEND
END SUB

'_____________________________
SUB parse(filename$,path$) STATIC
index%=1
    WHILE index% '<>0
        last%=index%
        index%=INSTR(index%+1,filename$,":")
    WEND
    IF last%=1 THEN  'there was no path
        path$=""
    ELSE             'divide filenamein$ into path$ and
filename$
        path$=LEFT$(filename$,last%)
        filename$=RIGHT$(filename$,LEN(filename$)-last%)
    END IF
END SUB
```


FORTH
FORUM
Macintosh Plus

Forth Forum
A SCSI Driver in Forth

Jörg Langowski
MacTutor Editorial Board
Grenoble, France
MacTutor Vol. 4 No. 2

"An experimental SCSI driver in Mach2"

This column almost didn't make it in time. Believe it or not, but while testing the SCSI driver of this article, my Quantum Q280 gave up. No, read on, it wasn't the software. (How could it be MY software, anyway?) The interface board let go, and there was my most recent version of the driver, on the disk, inaccessible, me dummy having made no backup for the last 2 days. Left with 'only' a 20 MByte Seagate drive, I was going through a non-trivial exercise on how to debug a SCSI driver on the same disk which is used for developing it. You cannot imagine the strange things that can happen when booting from a disk with a partially functioning driver; it sometimes forces you into doing absolute no-nos such as booting from a floppy and attaching the SCSI connector afterwards so you can re-install the driver without getting *la bombe* immediately. At least, the development cycle is slowed down appreciably, since every other time you'll have to boot from a floppy.

Anyway, here it is. The listing contains the Forth source of a SCSI driver which is more or less a translation of Apple's assembly source into Mach2. It is quite long, and I'm not going through all of it in detail, having explained the function of most of the routines in my last column. One added feature is error retry; if a SCSI error happens, the offending command is repeated up to ten times.

The driver takes 50% more space than the assembly version and is about half as fast. So, you may ask, why go through all this trouble? You might have noticed the title 'An experimental SCSI driver'. The code is meant for you to experiment with, add features - such as better error recognition and correction, multi-volume support, anything you may imagine. Or translate part of the routines into C or assembler. I just felt it was necessary to have a functioning example of a SCSI driver in some 'higher-level' language to make changes easier.

From the Forth point of view, I added a number of compiling words and glue routines that may be helpful for you in developing other things.

You already know the words **:XDEF**, **;XDEF**, and **XLEN** from the Hypercard article. For easier programming of drivers and desk accessories, I now added **:DA**, **;DA**, and **DALEN** which define the header of a driver, fill in the header after its definition is complete, and return the total length of the driver.

Since the SCSI driver code is jumped to at the beginning, it looks like other definition procedures from the outside, and its definition is therefore embedded between **:XDEF** and **;XDEF**. In its interior, the SCSI driver contains a real **DRVR**, with is bracketed between **:DA** and **;DA**. You recognize the glue code for the Open, Close, Control, Status and Prime routines at the end of the listing; the glue routines have been simplified by introducing macros for register save/restore and eventual jumps to the **JIODone** routine.

The Prime routine

Prime is the only one that has to be explained in more detail here since it has been modified with respect to the description given in the last issue. I added an error retry loop, and transfer of the SCSI data in variable size chunks. The code as printed takes 127 sectors on each read or write, that is, any amount of data larger than that will be transferred using several calls to the **SCSICommon** routine. This has been worked out empirically; larger transfer sizes would not work on the ST225N (which would result in files > 64K not being copied on a Finder copy). Your disk drive may allow larger sizes or need smaller ones; check this out if you like. You might also want to change the Read/Write Extended to a simple Read/Write command, or use a multi-transfer SCSI instruction block. The driver is certainly not generic and might need such adjustments to work on your disk.

SCSI Installation

The installation part has been changed slightly, and you'll have to replace parts of the installer code given in the last column. First, I changed the device and partition maps such that the driver starts at block 4 and the Macintosh volume partition at block 16. This is where the Apple Hard Disk installation program seems to expect it, therefore you can easily replace this driver with the Apple driver in case of problems, without haveing to reformat the disk. Furthermore, I added a small routine that will install the driver code in the system heap and call it so that it gets installed in the unit table; this is an easy way of testing the driver without writing it to disk. You can even install the same disk that you are using to write the program, in which case you will be left with two drive icons on the desktop, referring to the same disk. For testing, this is OK, only don't work too much in such an awkward configuration.

I hope this series of two articles has given you some ideas on how to write your own SCSI driver for your particular device. If someone comes up with a good examples for controlling a tape streamer - we're always open for contributions, as you know.

Feedback Dept.

Someone read my complaints about the missing editor in Mach2; a letter with a disk arrived here recently:

"Dear Mr. Langowski,

I'm afraid I'm not a fan of Forth [*Oh... JL*]. I like the idea of a stack-based language but there's an impenetrable jargon barrier... Besides, I just don't like typing shifted characters, like #, !, % and so on. So I stay home with Pascal and Assembler.

© The Definitive MacTutor, Vol. 4

On the other hand, I always at least glance at your column in MacTutor (as well as every other article that isn't Mac II-specific). In the October issue, you say, 'It is a shame that a powerful development system like Mach2 still lacks a reasonable integrated editor. At least multiple windows should be possible... My standard development system is Mach2.13 with its editor and Mockwrite, plus Edit under Switcher if required.' I'm stunned. The idea of using Mockwrite in a development system! The Mach2 editor must be really bad [*It's not bad for what it does, i.e., has no major bugs; but I admit its features are limited - JL*].

I had an editor problem myself, working with TML Pascal. TML won't really run under the Switcher; it demands too much memory. So if I make a syntax error, I have to transfer to the editor, which means quitting the compiler and launching the editor, fix the error, then quit the editor and re-launch the compiler. Then I can re-compile. Something of a pain, particularly since error-free code is somewhat rare, at least when I'm the coder! [*This sounds familiar. JL*] My solution: Afterthought, a desk accessory programming editor. Now I fix bugs without leaving the compiler.

My problem is solved, and yours may be as well. I'm sending you a copy of Afterthought. It will open large files (up to 8 Megabytes), two at a time. It has most of the features of MDS Edit - the most significant exception is Replace All. And it's reasonably fast. At the very least, it is better than Mockwrite!

Sincerely,

Clifford Story, Jimmy Mac Software, P.O.Box 957, Murfreesboro, Tennessee 37133."

Thank you, Clifford, for developing a product that - I guess - many of our readers have been waiting for.

The disk contained the Afterthought desk accessory, its manual, some update notes and examples, a demo version, and a demo version of Idealiner, an idea processor written by the same author. The prices are, by the way, extremely reasonable; $20 for the editor and $40 for the idea processor. To find out more about these products, send mail to the above address, or GEnie mail to CLIFF.

The second editor desk accessory product that I find worth mentioning is JoliWrite, written by Benoît Widemann from Paris, a small (32K limit) text editor that is extremely useful for working with bulletin board services. Often, when you prepare a text off line for a BBS message, you wish to be able to enter the text free-format and then format into lines of so and so many characters, with paragraph indentation and justification if desired. Also, over here it is often necessary to convert accentuated characters from Macintosh into ASCII representation and back, and in general you might wish to be able to remove/add line feeds, clip off numbers from line starts, etc. JoliWrite does all that, and in addition is one of the few products that supports Undo on all operations. By the time you read this, an English translation of JoliWrite (shareware, 120F/US$20) will hopefully be available on the Macintosh section of GEnie.

Listing 1: An experimental SCSI driver

```
( © 1987 J. Langowski / MacTutor )
only forth definitions
also mac also assembler

CODE SCALE
    MOVE.L   (A6)+,D0
    BMI.S    @1
    MOVE.L   (A6),D1
    ASL.L    D0,D1
    MOVE.L   D1,(A6)
    RTS
@1  MOVE.L   (A6),D1
    NEG.L    D0
    ASR.L    D0,D1
    MOVE.L   D1,(A6)
    RTS
END-CODE

: 4ASCII 0
    4 0 DO
    8 SCALE 0 WORD 1+ C@ +   LOOP
;

( *** compiler support words for external definitions *** )
: :xdef
    create  -4 allot
        $4EFA w, ( JMP )
        0 w,    ( entry point to be filled later )
        0 , ( length of routine to be filled later )
        here 6 - 76543
;

: ;xdef ( branch marker entry | - )
    marker 76543 <> abort" xdef mismatch"
    entry branch - branch w!
    here branch - 2+ branch 2+ !
;

: xlen 4 + @ ; ( get length word of external definition )

( *** driver header block *** )

 0 CONSTANT drvrFlags
 2 CONSTANT drvrdelay
 4 CONSTANT drvrEMask
 6 CONSTANT drvrMenu
 8 CONSTANT drvrOpen
10 CONSTANT drvrPrime
12 CONSTANT drvrCtl
14 CONSTANT drvrStatus
16 CONSTANT drvrClose
18 CONSTANT drvrname
50 CONSTANT DAlength

( *** compiler support words for DA and driver definitions ***
)
: :DA
    create  -4 allot
        here 87654 ( start of DA block, and marker )
        54 allot   ( length of block )
;

: ;DA ( DAstart marker Ropen Rprime Rctl Rstatus Rclose
        Rflags Rdelay Remask Rmenu Rname | - )
    marker 87654 <> abort" DA definition mismatch"
    Ropen   DAStart - DAStart drvrOpen + w!
    Rprime  DAStart - DAStart drvrPrime + w!
    Rctl DAStart - DAStart drvrCtl + w!
    Rstatus DAStart - DAStart drvrStatus + w!
    Rclose  DAStart - DAStart drvrClose + w!
    Rflags  DAStart drvrFlags + w!
```

```
    Rdelay  DAStart drvrDelay + w!
    Remask  DAStart drvrEmask + w!
    RMenu   DAStart drvrMenu + w!
    Rname count dup     DAStart drvrName + c!
        DAStart drvrName + 1+ swap
        dup 31 > if drop 31 then cmove
    here DAstart -  DAStart DAlength + !
;

: DAlen DAlength + @ ; ( get length word of external definition )

\ ------------------------
\ some macros needed in the driver
\ ------------------------

CODE xchg ( exchange word halves on top of stack )
    move.l (a6)+,d1
    swap.w d1
    move.l d1,-(a6)
    rts
END-CODE MACH

CODE min
    MOVE.L    (A6)+,D0
    CMP.L     (A6),D0
    BGE.S     @1
    MOVE.L    D0,(A6)
@1  RTS
END-CODE MACH

CODE shl ( data #bits )
    MOVE.L (A6)+,D0
    MOVE.L (A6),D1
    LSL.L  D0,D1
    MOVE.L D1,(A6)
    RTS
END-CODE MACH

CODE shr ( data #bits )
    MOVE.L (A6)+,D0
    MOVE.L (A6),D1
    LSR.L  D0,D1
    MOVE.L D1,(A6)
    RTS
END-CODE MACH

CODE w*
    MOVE.L (A6)+,D1
    MOVE.L (A6)+,D0
    MULS.W D1,D0
    MOVE.L D0,-(A6)
    RTS
END-CODE MACH

\ ------------------------
\ **** DA glue macros
\ ------------------------

$8FC CONSTANT JioDone

CODE DA.prelude
    LINK A6,#-512          ( 512 bytes of local Forth stack )
    MOVEM.L A0-A1,-(A7)  ( save registers )
    MOVE.L  A6,A3          ( setup local loop return stack )
    SUBA.L  #256,A3        ( in the low 256 local stack bytes )
    MOVE.L  A0,-(A6)       ( parameter block )
    MOVE.L  A1,-(A6)       ( device control entry )
    RTS          \ just to indicate the MACHro stops here
END-CODE MACH

CODE DA.epilogue
    MOVE.L  (A6)+,D0       ( return code )
    MOVEM.L (A7)+,A0-A1  ( restore registers )
    UNLK A6
    RTS
END-CODE MACH

CODE DA.JIODone
    MOVE.L  (A6)+,D0       ( return code )
    MOVEM.L (A7)+,A0-A1  ( restore registers )
    UNLK A6
    move.l  JIODone,A0
    movem.l d4-d7/a4-a6,-(a7)
    jsr     (a0)
    movem.l (a7)+,d4-d7/a4-a6
    RTS
END-CODE MACH

.trap    _newptr,sys,clr  $A71E

\ fields of device control entry
 4 CONSTANT dCtlFlags
 6 CONSTANT dCtlQHdr
16 CONSTANT dCtlPosition
20 CONSTANT dCtlStorage
24 CONSTANT dCtlRefNum
26 CONSTANT dCtlCurTicks
30 CONSTANT dCtlWindow
34 CONSTANT dCtlDelay
36 CONSTANT dCtlEMask
38 CONSTANT dCtlMenu

\ parameter block constants

0   CONSTANT  qLink    \ ptr to next queue entry[long word]
4   CONSTANT  qType    \ queue type [word]
6   CONSTANT  ioTrap       \ routine trap [word]
7   CONSTANT  ioTrap+1     \ read or write command
8   CONSTANT  ioCmdAddr    \ routine address [long word]
12  CONSTANT  ioCompletion    \ addr of completion routine
16  CONSTANT  ioResult     \ result code returned here
18  CONSTANT  ioNamePtr    \ pointer to file name string
22  CONSTANT  ioVRefNum    \ volume reference number
26  CONSTANT  csCode       ( type of control call )
28  CONSTANT  csParam      ( control call parameters )

\ MFS I/O Parameter Block
24 CONSTANT ioRefNum
26 CONSTANT ioVersNum
27 CONSTANT ioPermssn
28 CONSTANT ioMisc
32 CONSTANT ioBuffer
36 CONSTANT ioReqCount
40 CONSTANT ioActCount
44 CONSTANT ioPosMode
46 CONSTANT ioPosOffset
50 CONSTANT IOParamBlkSize

4ascii SDRV constant "sdrv
4ascii TFS1 constant "tfs1

\ Equates
\ My excuses for the format. This has been taken almost
\ 'as is' from Apple's SCSI driver example. - jl -

EQU verChar $34 \ version '4'
EQU SCSIZE  10  \ size of SCSI extended command

\ Equates for our storage (pointed to by DCtlStorage)
EQU Offset  0    \ [long] offset of starting sector
EQU MyDQE1      Offset+4
\ [20 bytes] drive queue element (with flags) for this drive
EQU MyDrvNum    MyDQE1+20
    \ [word] drive num (determined by scanning drive queue)
EQU NextAddr    MyDrvNum+2
```

```
    \ [long] ptr to current block buffer
EQU TickleFlag NextAddr+4
    \ [byte] Do we need to remind the system about this drive?
EQU BlindOK    TickleFlag+1
    \ [byte] Can we use blind reads?
    \ I left this in to keep the format the same.
    \ We don't need it since our driver does not read blind.
EQU SCmd    BlindOK+1
    \ [10 bytes] SCSI extended cmd Block /JL
EQU StatWord   SCmd+10        \
     [word] status and message bytes...
EQU MsgWord    StatWord+2
    \ [word] ... returned by SCSIComplete

EQU OurID      MsgWord+2 \ [word] our SCSI ID
EQU SCSIPseudo OurID+2
    \ [30 bytes] SCSI pseudo-code program
    \ - three instructions long
EQU SCSIPar1   SCSIPseudo+2 \ first SCSI code paramr (long)
EQU SCSIPar2   SCSIPar1+4\ 2nd SCSI code parameter (long)
EQU DiskVarLth SCSIPseudo+(SCSIZE*3)
    \ length of our locals . . .
EQU DQDrvSize  12

EQU realSizeMyDQE1+DQDrvSize+4

\ equates for CSParam offsets for our special control call
EQU DSCCmd     CSParam
    \ Ptr to SCSI command block
EQU DSCPseudo  DSCCmd+4  \ Ptr to SCSI pseudocode (if any
                         bytes to xfer)
EQU DSCBuffer  DSCPseudo+4 \ Ptr to buffer for transfer (if
                           any)
EQU DSCSize    DSCBuffer+4   \ Size of transfer, signed (+ if
                             read, - if write)
EQU DSCTicks   DSCSize+4 \ Tick count we're willing to wait
                         for completion
EQU DSCCmdSize DSCTicks+4 \ (word) Size of command block we're
                          sending (usually 6)

EQU KillCode   1
EQU VerifyCode 5
EQU FormatCode 6
EQU EjectCode  7
EQU IconCode      21
EQU AccRun     65
EQU SCSICode   77
       \ our own special code (defined above)

EQU ControlErr -1
EQU StatusErr     -1
EQU ParamErr      -50
EQU nsDrvErr      -56
EQU nsVErr     -35
EQU ioErr         -36

EQU dNeedTime  $DFFF
       \ to clear bit 5 of high byte in drvrFlags

EQU DiskInsertEvt 7
EQU SysEvtMask $144
EQU UTableBase $11C
EQU DrvQHdr    $308
EQU QHead      $2

EQU DQDrive    6
EQU DQRefNum   8
EQU DQFSid     10

EQU PDSig      0
EQU PDSigWord  $5453
EQU PDFSID     8
EQU PDLen      12

\ ____________________________________________________
\ The code starts here.
\ ____________________________________________________
:XDEF    ScsiDisk
    \ compiles a jump to the install code at the end
    \ which will be resolved at the end of the definition.
:DA DiskDrvr
       \ this word provides the driver header structure
       .ALIGN
\ Q200 Icon, as given by Quantum
\ If you find this 'snail' ugly, feel free to change it ...
\  J.L.
header  SCSIIcon
DC.L $00000000  DC.L $00000000
DC.L $00000000  DC.L $000FF000
DC.L $003FFC00  DC.L $00FFFF00
DC.L $01FFFF80  DC.L $03F81FC0
DC.L $07E007E0  DC.L $07C003E0
DC.L $0F8001F0  DC.L $0F0000F0
DC.L $1F0000F8  DC.L $1E000078
DC.L $1E000078  DC.L $1E000078
DC.L $1E000078  DC.L $1E000078
DC.L $1E000078  DC.L $1F0000F8
DC.L $0F0000F0  DC.L $0F8001F0
DC.L $07C003E0  DC.L $07E007E0
DC.L $03F80000  DC.L $01FFFFF0
DC.L $00FFFFF8  DC.L $003FFFF8
DC.L $000FFFF8  DC.L $00000000
DC.L $00000000  DC.L $00000000

DC.L $00000000  DC.L $00000000
DC.L $000FF000  DC.L $003FFC00
DC.L $00FFFF00  DC.L $01FFFF80
DC.L $03FFFFC0  DC.L $07FFFFE0
DC.L $0FFFFFF0  DC.L $0FFFFFF0
DC.L $1FFFFFF8  DC.L $1FFFFFF8
DC.L $3FFFFFFC  DC.L $3FFFFFFC
DC.L $3FFFFFFC  DC.L $3FFFFFFC
DC.L $3FFFFFFC  DC.L $3FFFFFFC
DC.L $3FFFFFFC  DC.L $3FFFFFFC
DC.L $1FFFFFF8  DC.L $1FFFFFF8
DC.L $0FFFFFF0  DC.L $0FFFFFF0
DC.L $07FFFFF0  DC.L $03FFFFF8
DC.L $01FFFFFC  DC.L $00FFFFFC
DC.L $003FFFFC  DC.L $000FFFFC
DC.L $00000000  DC.L $00000000

\ Our "Where:" string
       DC.B    11
       DC.B    'Q200 (SCSI)'
       .ALIGN
\ SCSI handler glue routines
CODE SCSIReset ( - result code )
     CLR.W -(A7)
     MOVE.W #0,-(A7)
     _SCSIDispatch
     MOVE.W (A7)+,D0
     EXT.L   D0
     MOVE.L D0,-(A6)
     RTS
END-CODE

CODE SCSIGet ( - result code )
     CLR.W -(A7)
     MOVE.W #1,-(A7)
     _SCSIDispatch
     MOVE.W (A7)+,D0
     EXT.L   D0
     MOVE.L D0,-(A6)
     RTS
END-CODE
```

```
CODE SCSISelect ( TargetID - SCSIErrorResult )
    MOVE.L (A6)+,D0
    CLR.W -(A7)
    MOVE.W D0,-(A7)
    MOVE.W #2,-(A7)
    _SCSIDispatch
    MOVE.W (A7)+,D0
    EXT.L  D0
    MOVE.L D0,-(A6)
    RTS
END-CODE

CODE SCSICmd ( buffer count - SCSIErrorResult )
    MOVE.L (A6)+,D0
    MOVE.L (A6)+,D1
    CLR.W -(A7)
    MOVE.L D1,-(A7)
    MOVE.W D0,-(A7)
    MOVE.W #3,-(A7)
    _SCSIDispatch
    MOVE.W (A7)+,D0
    EXT.L  D0
    MOVE.L D0,-(A6)
    RTS
END-CODE

CODE SCSIComplete ( waitTicks mess stat - SCSIErrorResult )
    CLR.W -(A7)
    MOVE.L (A6)+,-(A7)
    MOVE.L (A6)+,-(A7)
    MOVE.L (A6)+,-(A7)
    MOVE.W #4,-(A7)
    _SCSIDispatch
    MOVE.W (A7)+,D0
    EXT.L  D0
    MOVE.L D0,-(A6)
    RTS
END-CODE

1 CONSTANT SCInc   2 CONSTANT SCnoInc
3 CONSTANT SCAdd   4 CONSTANT SCMove
5 CONSTANT SCLoop  6 CONSTANT SCNop
7 CONSTANT SCStop  8 CONSTANT SCComp

\ ------------------------
\ main driver routines start here
\ ------------------------

: SCSICommon
        \ written to emulate the SCSICommon
        \ routine in Apple's example
        \ as closely as possible.
    ( pseudo cmdblock ourVars ticks bytes cmdsize
        | writing mess stat - result )

    SCSIGet 0= IF
        ourVars ourID + w@
        SCSISelect 0= IF
            cmdBlock cmdSize SCSICmd 0= bytes AND IF
                pseudo bytes 0< \ bytes <0 if writing
                IF (call) SCSIWrite drop
                ELSE (call) SCSIRead drop THEN
\ Note: Your system may be able to support blind transfers.
\ Here is the place to experiment with such things -
            THEN
            ticks ^ mess ^ stat SCSIComplete
            0= IF
                stat $FF AND IF ioErr
                      ( there was an SCSI error )
                    ELSE 0 ( successful completion ) THEN
            ELSE ( complete unsuccessful ) ioErr
            THEN
        ELSE ( select unsuccessful ) ioErr
        THEN
    ELSE ( get unsuccessful ) ioErr
    THEN
;

: DiskClose ( parblk dce | - result )
    0 ( result code = OK ) ;

: diskControl ( parblk dce | ourVars - result )
    dce DCtlStorage + @ -> ourVars
    parblk csCode + w@
    CASE
        killCode  OF 0 ENDOF
        verifyCodeOF 0 ENDOF
        formatCodeOF 0 ENDOF

        ejectCode OF
            ourVars MyDrvNum + w@
                \ check drive # in request
            parblk  IOVRefNum + w@ =  \ the same?
          IF
            SysEvtMask w@ IF ( we're not at boot time )
               DiskInsertEvt
               MyDrvNum ourVars + w@
               (call) PostEvent drop
               ELSE ( boot time )
               1 ourVars tickleFlag + c!
               ( drive will be remembered after boot )
               THEN
            controlErr
          ELSE nsDrvErr
          THEN
          ENDOF

        iconCode OF ['] SCSIIcon parblk csParam + !
          0 ENDOF

        accRun OF
          ourVars tickleFlag + c@
          ourVars offset + @ 0= not
            ( we have a good partition )
            AND
            IF
            DiskInsertEvt
            MyDrvNum ourVars + w@
            (call) PostEvent drop
            THEN
          0 dce DCtlDelay + w!
          dce DCtlFlags + dup w@
            dNeedTime AND swap w! ( clear flag )
          0 ourVars tickleFlag + c!
          0 ENDOF

        scsiCode OF
          parblk dup DSCPseudo + @
                 dup DSCCmd + @
                   ourVars
                 dup DSCTicks + @
                 dup DSCSize + @
                   DSCCmdSize + w@
          SCSICommon
          ENDOF

        ( otherwise )
        controlErr
    ENDCASE
;

: DiskStatus ( parblk dce | - result ) statusErr ;

CODE GetSysPtr
    move.l  (a6)+,d0
    _newptr,sys,clr
```


© The Definitive MacTutor, Vol. 4

```
    move.l  a0,-(a6)
    rts
END-CODE

CODE AddDrv ( dqe refnum drv# | - )
    move.l  (a6)+,d0
    move.l  (a6)+,d1
    swap.w  d0
    move.w  d1,d0
    move.l  (a6)+,a0
    _AddDrive
    rts
END-CODE

: DiskOpen ( parblk dce |
        ourVars thisQElem driveNum dqe SCSIprog - result )

    DiskVarLth GetSysPtr dup
        \ get memory for local variables
    -> ourVars dce DCtlStorage + !
        \ and store pointer to it
    100 5 DO            \ find unused drive #
        DrvQHdr QHead + @ -> thisQElem  \ scan queue
        BEGIN thisQElem 0= IF i leave THEN
              \ end of queue? we have a good number
           thisQElem DQDrive + w@
           i <> WHILE
              \ keep scanning as long as # is not in use
           thisQElem ( QLink + ) @ -> thisQElem
        REPEAT
    LOOP -> driveNum
    driveNum ourVars myDrvNum + w!
              \ remember drive # in local vars
\ ( following text taken from Apple's generic driver )
\ Add a drive to the drive queue. First, some fun facts:
\ The drive queue element starts four bytes before the DQEPtr!
\ These four bytes contain "hardware-locked", "ejectable", and
\ "disk-in-place" info.
\
\ Not As Interesting But Still True: HFS supports volumes
\ >32MBytes, but since the dqDrvSize field in the DQE is only
\ a word, the Software Gurus had to resort to bizarre sorcery:
\ If the qType field (formerly unused in DQE's) is 1, the word
\ following the dqDriveSize field is assumed to be the high-
order
\ word of a LongInt block count! (dqDriveSize is still the
low-
\ order word). It works even if the size doesn't require both
\ words, so we always do it this way.
\
\ See: Tech Note #36.

    ourVars MyDQEl + 8 over w!
        \ set non-ejectable and clear the rest
        2+ 0 over w! 2+  -> dqe
        \ this is the real start of the DQElem
    1 dqe qType + w!  \ large vol queue type
    0 dqe dqDrvSize + !   \ no size yet
    0 dqe dQFSID + w!     \ normal file system
    dqe
    dce DCtlRefNum + w@
    driveNum    AddDrv \ add drive to queue

    \ now set up SCSI pseudo program in driver's local vars

    ourVars SCSIPseudo +  -> SCSIprog
    scnoinc SCSIprog w!
    scstop  SCSIprog scsize + w!
    0 \ result code = good
;

: DiskPrime ( parblk dce |
    ourVars sectors bytes start size r/w sect transferred
error
          - result )

    dce dCtlStorage + @ -> ourVars  \ setup local var pointer
    1 ourVars TickleFlag + c!

        \ convert byte count into number of sectors
    parblk IOReqCount + @ 9 shr $1FFFFF AND -> sectors
        \ convert starting position into sector number
    dce dCtlPosition + @ 9 shr $1FFFFF AND -> start

    ourVars realSize + @ xchg -> size \ get drive size
    start sectors + size 1+ < IF ( valid request )
        0 -> transferred
        ourVars ( offset + ) @ +> start
              \ offset by start of partition

        parBlk IOTrap+1 + c@ 3 = ( is this a write command? )
        IF  -1 -> r/w $2A00     ( SCSI extended write )
        ELSE 1 -> r/w $2800 ( SCSI extended read )
        THEN
           ourVars SCmd + w!\ put the command away

        BEGIN ( transfer loop )

\ If you have problems getting the SCSI transfer to work
\ with your particular disk, try changing the number of
\ sectors transferred on each call ( 127 here )
\ or change the read/write extended to a normal read/write.
    \ Note that in that case you'll have to change the command
    \ block setup as well.

        127 sectors min -> sect
        transferred +> start
        parblk IOBuffer + @ transferred 9 shl +
           ourVars SCSIPar1 + !
        sect 9 shl dup -> bytes
           ourVars SCSIPar2 + !  \ set # of bytes

        start ourVars SCmd + 2+ !
              \ set starting position in command block
        bytes 2/ ourVars SCmd + 6 + !   \ set # of sectors

        IOErr ( preset, if loop with retry is unsuccessful )
        10 0 DO ( retry max 10 times )
        ourVars SCSIPseudo +
           ourVars SCmd +
           ourVars 60 r/w 10
           SCSICommon 0= IF drop 0 leave THEN
        1 (call) sysbeep \ just for debugging,
           \ beeps if SCSI did not complete successfully
        LOOP -> error
        -127 +> sectors
        sectors 1- 0<
        UNTIL ( transfer loop )
        error dup 0= IF
           parBlk IOReqCount + @ -> bytes
           bytes parBlk IOActCount + !
              \ we transferred the # of bytes requested
           bytes dce DCtlPosition + +!
        THEN
    ELSE IOErr
    THEN
;

CODE DrvrInst ( unitNum | - )
    move.l  (a6)+,d0
    not.w   d0
    _DrvrInstall
    rts
END-CODE

CODE DrvrRem ( unitNum | - )
```

```
    move.l  (a6)+,d0
    not.w   d0
    _DrvrRemove
    rts
END-CODE

CODE openMe ( drvrName | result - )
    \ allocates a parameter block on the A7 stack and calls
    \ the _open trap. This is easier to do in assembly -
    moveq.l #(IOParamBlkSize/2)-1,d0
@1  clr.w   -(a7)
    dbra d0,@1
    move.l  a7,a0
    move.l  (a6)+,IONamePtr(a0)
    _Open
    add.w   #IOParamBlkSize,a7
    move.l  d0,-(a6)
    rts
END-CODE

: RealInstall
\ This routine is called by the system boot code with
\ the SCSI ID of the disk in D5 and a pointer to its
\ partition map in A0. We therefore need some special glue
code.
\ Note that Mach2 allows to do the stack parameter / local
\ variable declaration after this glue code without any
problems
    LINK A6,#-512      ( 512 bytes of local Forth stack )
    MOVEM.L A2-A6/D2-D7,-(A7)      ( save registers )
    MOVE.L  A6,A3      ( setup local loop return stack )
    SUBA.L  #256,A3       ( in the low 256 local stack bytes )
    MOVE.L  A0,-(A6)      ( partition table pointer )
    MOVE.L  D5,-(A6)      ( SCSI ID )

    { partition ID | unitNum hdce dce ourVars pt - }

    ID 32 + -> unitNum
    unitNum DrvrInst  \ allocate DCE and install it
    unitNum 4 w* UTableBase @ + @ -> hdce \ dce handle
        hdce @ -> dce \ get dce pointer
    ['] DiskDrvr dce ( DCtlDriver + ) !
        \ put pointer to driver into dce
    ['] DiskDrvr drvrFlags + w@
        dce DCtlFlags + w!
        \ move driver flags, RAMbase should be cleared
    0 dce DCtlDelay + w!      \ no time needed yet
    ['] DiskDrvr drvrEMask + @
        dce DCtlEMask + !    \ move event mask and menu

    ['] DiskDrvr drvrName + openMe  \ try to open this driver
        IF ( not OK ) unitNum DrvrRem
           ['] Scsidisk (call) DisposPtr
           bra @1 \ exit hack.
           \ This is the Mach2 equivalent of the
           \ Ugly Goto Statement in Pascal.
           \ Sorry, but it is so much easier this way...
        THEN

    hdce @ -> dce
           \ deref this handle again, may have changed
    dce dCtlStorage + @ -> ourVars
    ID ourVars ourID + w!
    partition IF
    \ well, we should have a non-NIL partition at least...
    partition ( PDSig + ) w@ PDSigWord = IF
    \ and it should be a Macintosh one. The NEW Apple drivers
    \ have a different sig word and DPM format that you
    \ might want to take into account here (see text).
        partition 2+ -> pt
        BEGIN
        pt PDFSID + @ ?dup WHILE
           \ otherwise no good partition found
           "tfs1 =
              IF ( correct file system ID )
                 pt @ ourVars Offset + !
                 pt 4 + @ xchg
                    ( long drive size, hi word <-> lo word )
                    ourVars realSize + !
                 SysEvtMask w@ 0= IF \ we're booting
                    dce dCtlFlags + dup w@
                       $2000 OR swap w!
                       ( set dNeedTime flag )
                    1 dce dCtlDelay + w!
                    1 ourVars TickleFlag + c!
                    THEN
              THEN
           12 +> pt
           REPEAT
        THEN THEN

@1  UNLK A2 \ which was used for local variables
    MOVEM.L (A7)+,A2-A6/D2-D7   ( restore registers )
    UNLK A6
    RTS
\ we stop here; the rest will be inaccessible junk (4 bytes).
;


: DrOpen  DA.prelude DiskOpen DA.epilogue ;
: DrClose DA.prelude DiskClose DA.epilogue ;
: DrCtl   DA.prelude DiskControl DA.JIODone  ;
: DrStatus DA.prelude DiskStatus DA.JIODone ;
: DrPrime DA.prelude DiskPrime DA.JIODone ;

' DrOpen ' DrPrime ' DrCtl ' DrStatus ' DrClose
$6F00 0 0 0 ( flags delay mask menu )
" .SCSIfth" ( name, MUST start with a period )
;DA

' RealInstall ;XDEF

\ ------------------------
\ The following routines are to be added or replaced in the
\ installer program from the previous column. Included is an
\ installer that will directly move the Forth code to disk,
without
\ going through a resource, and some code to install the
driver
\ in memory for testing without writing it to the disk. The
\ DDM and DPM definitions have been changed somewhat
\ to accommodate the larger driver, and to have the partition
start
\ at the same place that Apple's new SCSI driver expects it
(so
\ that you can replace the Forth driver easily by a new Apple
\ driver in case you are fed up with this hack)
\ Good luck. - JL -
\ ------------------------

hex

: create.ddm
    ddm 200 0 fill
    4552 ddm w!
    read.cap ddm 2+ w! ( block size )
             ddm 4 + ! ( # of blocks )
    0 ddm 8 + w! ( device type )
    0 ddm A + w! ( device ID )
    10 ddm C + !  ( first data block )
    1 ddm 10 + w! ( one driver to follow )
    4 ddm 12 + ! ( driver start block )
    A ddm 16 + w! ( driver is 10 blocks long )
    1 ddm 18 + w! ( and runs on Macintosh =1 )
;
: create.dpm
```

```
    dpm 200 0 fill
    5453 dpm w!
    10 dpm 2+ ! ( starting block of partition )
    read.cap drop 10 - dpm 6 + ! ( # of blocks )
    "tfs1 dpm A + !   ( TFS1 signature )
    0 dpm E + !
;
decimal
: read.ddm
    0 read.blk 2+ w! 0 read.blk 4 + c!
    1 read.blk 5 + c!
    120 read.blk myDisk @ ddm 512 doscsi.r
    2drop
;
: read.dpm
    0 read.blk 2+ w! 1 read.blk 4 + c!
    1 read.blk 5 + c!
    120 read.blk myDisk @ dpm 512 doscsi.r
    2drop
;
: write.ddm
    0 write.blk 2+ w! 0 write.blk 4 + c!
    1 write.blk 5 + c!
    120 write.blk myDisk @ ddm 512 doscsi.w
    2drop
;
: write.dpm
    0 write.blk 2+ w! 1 write.blk 4 + c!
    1 write.blk 5 + c!
    120 write.blk myDisk @ dpm 512 doscsi.w
    2drop
;
: get.sdrv ( | length - length )
    ['] scsidisk dup
    xlen dup -> length driver.block swap cmove
    length
;
: write.sdrv ( length | sectors )
    0 write.blk 2+ w! 4 write.blk 4 + c!
    length 512 / 1+ dup write.blk 5 + c! -> sectors
    120 write.blk myDisk @ driver.block
        sectors 512 * doscsi.w
    cr ." Driver written. Stat, Mess = " . .
;

: dmp ( block# | - ) ( for easy testing of SCSI disk contents
)
    0 read.blk 2+ w! block# read.blk 4 + c!
    1 read.blk 5 + c!
    120 read.blk myDisk @ ddm 512 doscsi.r
    2drop
    ddm 20 dump
;

.TRAP   _newptr,sys     $A51E
$308 CONSTANT DQHeader
6 CONSTANT QTail

VARIABLE syshp.drvr

: install.driver ( | dstart dlength dbytes pointer - )
    read.ddm
    ddm 18 +  @ -> dstart
    ddm 22 + w@ -> dlength
    cr ." Driver starts at sector " dstart .
    ."  and is " dlength . ."  sectors long."
    dlength 512 * -> dbytes
    dstart 256 /mod read.blk 2+ w! read.blk 4 + c!
    dlength read.blk 5 + c!
    120 read.blk myDisk @ driver.block dbytes doscsi.r
    cr ." Driver read; stat, mess = " . .
    dbytes  MOVE.L (A6)+,D0
    _newptr,sys ( get memory block in system heap )
    MOVE.L A0,-(A6)   -> pointer
    pointer
    IF   driver.block pointer dbytes cmove
        pointer syshp.drvr !
    ELSE ." Not enough system heap for installation." cr
    THEN
;
CODE call.driver
    MOVE.L D5,-(A7)
    MOVE.L (A6)+,D5
    MOVE.L (A6)+,A0
    execute
    MOVE.L (A7)+,D5
    RTS
END-CODE

: mount.scsi
    install.driver
    read.dpm
    SysEvtMask @
    0 SysEvtMask !
    syshp.drvr @ dpm myDisk @ call.driver
    SysEvtMask !
;
: zero.scsi
    DQHeader qTail + @ dQDrive + w@ ( drive # found )
    cr ." Do you want to zero the directory of drive # "
        dup . ."  ? "
    yesno if " JL's Hard Disk" call DIZero
            cr ." Result code = " . cr
    then
;
: mount
    cr ." Looking for SCSI devices..."
    get.disk
    cr ." SCSI drive found at address " myDisk @ .
    cr show.cap
    cr ." format disk? "
    yesno IF
        cr ." Do you REALLY want to erase this SCSI disk? "
        yesno IF cr ." Reformatting disk... "
                format
            THEN
        THEN
    modenoattn
    create.ddm create.dpm
    write.ddm  write.dpm
    cr ." Device and partition descriptor maps written. "
    get.sdrv
    cr ." Writing driver ... "
    write.sdrv
    mount.scsi
    zero.scsi
;
: install.mem ( | dbytes pointer - )
    get.sdrv
    ['] scsidisk xlen dup -> dbytes
    MOVE.L (A6)+,D0
    _newptr,sys ( get memory block in system heap )
    MOVE.L A0,-(A6)   -> pointer
    pointer
    IF   driver.block pointer dbytes cmove
        pointer syshp.drvr !
    ELSE ." Not enough system heap for installation." cr
    THEN
;
: mount.mem
    install.mem
    read.dpm
    SysEvtMask @
    0 SysEvtMask !
    syshp.drvr @ dpm myDisk @ call.driver
    SysEvtMask !
;
```

Forth Forum
Time Manager Reminder DA

Jörg Langowski
MacTutor Editorial Board
Grenoble, France
MacTutor Vol. 4 No. 3

Jörg Langowski is a bio-chemical engineer for a french concern and is a founding and current board member of MacTutor. He can be reached at EMBL, c/o I.L.L. 156x, Grenoble, Cedex, France F-38042. We encourage European authors to contact Jörg directly about submissions to MacTutor.

"Using the Time Manager"

One thing I always wanted to have on my Mac is some utility that reminds me of an appointment, no matter what I'm doing on the machine at any given time. On our VAX, for example, there is this great program called Reminder which sits in the background and will beep at you when your next appointment arrives.

In the new system (128K ROMs and up) there exists a set of routines called the Time Manager. I thought it would be a good idea to show you their usage in developing a Reminder utility similar to the one on the VAX. In the process, I'll show you how to work with modeless dialogs from a DA, how to call memory manager dependent traps from a time manager task, and some more.

The Time Manager

The Time manager is used to execute a task at a predetermined time. Its routines are documented in IM Vol. 4. All time manager tasks are described by entries in the time mamager queue, where each element has the following format:

```
header TMtask 14 allot
0 CONSTANT qLink
4 CONSTANT qType
6 CONSTANT tmTask
10 CONSTANT tmCount
```

That means, the queue entry looks just like a regular queue element, with two 4-byte fields, *tmTask* and *tmCount*, relevant to our task. *tmTask* contains a pointer to the routine to be executed, and *tmCount* is used by the system to count down the elapsed time. When it has reached zero, the task at *tmTask* will be executed.\

Three routines are provided by the time manager: *InsTime*, called with a pointer to the queue element in A0, will insert this queue element into the time manager queue. *PrimeTime*, with the queue element pointer in A0 and a 32-bit *count* in D0, will schedule the routine specified in that queue element to be executed after *count* milliseconds have elapsed. *RmvTime*, with a pointer to the queue element in A0, will remove the queue element. The Mach2 interface to these routines is given at the beginning of listing 1.

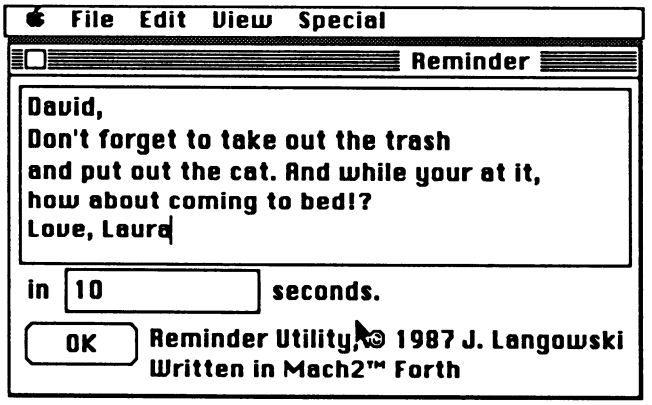


Fig. 1 Our Reminder DA lets Mac Widows post alerts for late night programmers!

Programming the Reminder utility

The strategy to follow seems very simple: we create a desk accessory that allows us to enter a message text and a delay time, that desk accessory will then setup a queue element with a pointer to a routine that displays an alert box containing that string.

Thinking about it, programming such a utility is not quite so simple. First of all, the routine passed to the time manager for execution cannot be part of the desk accessory. We'd like to enter appointments that are due in several hours or days (32 bits of milliseconds, unsigned, give a maximum possible delay of 49.7 days), and we'll most probably have left the application where the desk accessory was started by then. Under MultiFinder this doesn't matter, but under Finder, the desk accessory would have

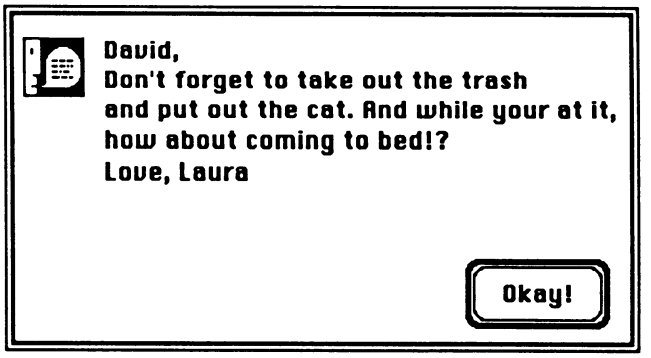


Fig.2 After time has elapsed, an alert pops up!

© The Definitive MacTutor, Vol. 4

been closed, leaving the task pointer point to anything but the desired code.

My first thought was to use a 'stick-around' desk accessory that would automatically restart itself after an involuntary close, like I described earlier. But it is actually much easier to put a copy of the routine to be executed into a small block of system heap, leaving it independent of the DA. Then, one has to take caution that the routine, once executed, will remove its own queue entry and dispose of its own memory.

The second problem is that a time manager task may be executed at *any* time, interrupting whatever else is being done at that moment. For the same reason as in the case of VBL tasks, we may not call any routines from the time manager task that can move things around in the heap. If we did, there would be a chance that the interrupt occurs right in the middle of a memory manager operation, or while a handle hangs around dereferenced somewhere.

The time manager task must therefore be rather simple. I use the same trick as I described for the 'stick-around' DA, patching SystemTask from the scheduled routine. The next time SystemTask is called (which should happen rather soon in any benign application) the patch routine will be executed, which then can do the more involved stuff such as drawing an alert box.

Now look at the first part of the example: the routine that is executed by the time manager is contained between the markers **mytask** and **mytask.end**. The first bytes of this block contain the queue element. The name string 'Reminder' follows for debugging purposes. Some local variable space is reserved, where the original SystemTask trap address, the alert ID and the alert message can be kept. The SystemTask patch routine **alertMe** follows. It will display a note alert with the message in parameter ^0, unlock the alert, remove the time manager queue element and dispose of the memory allocated for itself.

alertMe is patched into SystemTask by the following routine, **wakeMe**, which is the one first executed by the time manager. A pointer to **wakeMe** will be contained in the *tmTask* field of the queue element at the beginning of the block. Since we do not know yet where this routine will be located, we'll have to install the pointer after moving the code into the system heap.

As you see, the code between **mytask** and **mytask.end** is completely self-contained (as long as the ALRT resource can be found). We may therefore move any number of these little code blocks into the system heap and install their queue elements using the time manager; each will be executed at its scheduled time and display its little message. Appointment 'objects', so to say.

The desk accessory

The DA is used to install the tasks. It is a very simple DA which just draws up a modeless dialog box with two editable text items. The dialog window will have the driver reference number in its *windowKind* field, so that the system will handle mouse down events in the close box and drag region. Only when the event is passed through to the DA, *windowKind* will be temporarily set to 2 (dialog window), so that IsDialogEvent and DialogSelect work correctly.

When the OK button in the dialog box has been clicked, the dialog handler will be invoked. This routine converts the string in item 4 to a number, the delay in seconds, and gets the message to be displayed after this delay (item 3). These parameters are passed to **install.wakeup**, together with the alert ID. **install.wakeup** gets the wakeup routine described above from the resource file and installs *a copy of it* in the system heap. It saves a pointer to the wakeup routine in the *tmTask* field of the queue element, stores the alert ID and the message string at their appropriate positions and then schedules the routine for execution by calling **InsTime** and **PrimeTime**.

Things missing

The desk accessory described here is already quite useful 'as is'. However, one could include some improvements that I leave as an exercise for you (or me, for that matter):

- Input of the appointment time in standard date format.
- Keeping a list of appointments that can be displayed and edited.
- Saving this list to a file every time it is changed; this file is then checked on system startup and the pending appointments re-scheduled automatically.

I wish you good luck with those experiments; now, some mail that I recently received.

Feedback Dept.

"Dear Jörg,

I read your recent column in MacTutor with great interest and would like to obtain the addresses for Serge Rostan and also for Winsoft. *[Serge Rostan, "TechnoPro", rue Faraday, F-78180 Montigny le Bretonneux, France, phone (33) 1 30 45 26 62, and Winsoft, 34, boulevard de l'Esplanade, F-38000 Grenoble, France, phone (33) 76 87 56 01 - JL].*

We are in the process of completing a book on fonts for the Macintosh for European and many other non-Roman languages. It will also show specimens of over 2000 bit-mapped and several hundred PostScript fonts.

If you can suggest any font vendors that do not advertise in US Macintosh magazines, it would be very helpful *[not off the top of my head, but I'll look around - JL].*

As you may know, Apple refuses to sell European keyboards in the US, even to certified developers. Users here must cultivate acquaintances overseas and buy them indirectly. Do you know whether Apple tries to prevent the export of such keyboards from Europe? We would like to be able to list a commercial source for such keyboards in our book, but do not want to cause difficulties for anyone there. If they are readily available for export, we would appreciate the name of a recommended source.

[You address a problem that I've encountered myself. There seems no way for Macintosh developers in France to get other than French keyboards through the developer program. How-

ever, there is at least one Apple dealer in Grenoble who sells the US keyboard on request; we have several international research institutions here, so there's a market. I'll send you addresses of some Apple dealers in Germany and France that you might be able to order from.

I don't think there are any export restrictions imposed by Apple on non-US keyboards into the US (unless you re-sell them to Colonel Kh...). The problem most probably is the bureaucracy that inevitably builds up in a large corporation, even one like Apple that for a while tried to maintain some 'non-conformistic hacker' kind of image. Which it is trying to forget at a rapid rate, I presume. Someone, it seems, must have set a guideline that in country X only systems localized for X will be sold, with few exceptions. The only way I get the most recent US system updates is through US sources, too. Funniest of all: The MacII system that I finally got a week ago (yes, I love it), has English documentation, US system disks (but 4.1, not 4.2, which is not officially released at the time I write this), but no way can I get a US keyboard. Probably the people at Apple France just wanted to be nice and ship me as many US parts as possible, but then they had only French keyboards. What a nuisance to have all the numbers on the top row...]

Lastly, Apple also tries to limit the availability of any Script Manager code or fonts it has developed to the country for which it was developed! This seems strange given that the Macintosh is touted as a Multinational machine. Apple has made Kanji and Arabic available through APDA, but feels that they are sufficient for the testing purposes intended. They have also developed a Chinese, a Hebrew and a Greek. The Chinese is distributed from Hong Kong and the Hebrew and Greek are distributed from Paris. I have written to the Paris office to no avail after a conversation with Mark Davis at Apple, the creator of the Script Manager. Do you know of a way for someone in the US to get these fonts from Paris? Actually, I am more interested in finding out the arrangement of characters in the coding table than I am in the font, but the exact status of 'dead' keys might be hard to discern accurately without actually having the font in hand.

Best regards,

Tim Ryan, SourceNet, Santa Barbara, CA "

[You address a very timely issue, that's why I have your letter printed immediately without having the answers on hand yet. I'll forward your mail with my comments to Apple France. I'm actually quite optimistic that you can get the Script Manager versions that you want. JL]

One comment to the letter from Peter Adamson, MT V4#1, p.11: The Pascal equivalent to the Mach2 PAUSE is actually WaitNextEvent; much like PAUSE under Mach2, WaitNextEvent will transfer control to the next MultiFinder task under certain circumstances. If you have a very long event loop, you may try to intersperse WaitNextEvents with event masks of zero, so that they'll always return a null event. That should transfer control to the next task under MultiFinder. Only the crucial WaitNextEvent - the one with the BIG case statement behind it - would be called with a non-zero event mask. This is all untested, so don't blame me if it doesn't work.

See you next month.

Listing 1: Appointment reminder using the time manager

```
\ ***** Time manager example - a 'reminder' utility
\ J. Langowski December 87
\
\ Strategy: write a driver that sets up a dialog which allows
\ to enter a time & message to display after that time. After
\ the appointment has been entered, the driver sets up a
\ time manager call for that appointment.
\ The time manager routine installs a SystemTask trap
\ patch which at the next occasion will draw an alert box
\ containing the message to be displayed.
\
\ Note that we have to use the patch rather than calling
\ the alert routine directly from our time manager task,
\ since we can't be sure we're not in the middle of a
\ memory manager operation when it is called.
\

only forth also assembler also mac

CODE InsTime ( tmTaskPtr | - )
     MOVE.L (A6)+,A0
     _InsTime
     RTS
END-CODE MACH

CODE PrimeTime ( tmTaskPtr count | - )
     MOVE.L (A6)+,D0
     MOVE.L (A6)+,A0
     _PrimeTime
     RTS
END-CODE MACH

CODE RmvTime ( tmTaskPtr | - )
     MOVE.L (A6)+,A0
     _RmvTime
     RTS
END-CODE MACH

4ascii MENU constant "menu
4ascii PROC constant "proc

\ *** compiler support words for external definitions ***
: :xdef
     create  -4 allot
        $4EFA w, ( JMP )
        0 w,  ( entry point to be filled later )
        0 ,   ( length of routine to be filled later )
        here 6 - 76543
;

: ;xdef ( branch marker entry | - )
     marker 76543 <> abort" xdef mismatch"
     entry branch - branch w!
     here branch - 2+ branch 2+ !
;

: xlen 4 + @ ; ( get length word of external definition )

( *** driver header block *** )

 0 CONSTANT drvrFlags
 2 CONSTANT drvrdelay
 4 CONSTANT drvrEMask
 6 CONSTANT drvrMenu
 8 CONSTANT drvrOpen
10 CONSTANT drvrPrime
12 CONSTANT drvrCtl
```

```
14 CONSTANT drvrStatus
16 CONSTANT drvrClose
18 CONSTANT drvrname
50 CONSTANT DAlength

\ *** compiler support words for DA and driver definitions
: :DA
     create  -4 allot
        here 87654 ( start of DA block, and marker )
        50 allot   ( length of block )
;

: ;DA ( DAstart marker Ropen Rprime Rctl Rstatus Rclose
        Rflags Rdelay Remask Rmenu Rname | - )
     marker 87654 <> abort" DA definition mismatch"
     Ropen       DAStart - DAStart drvrOpen + w!
     Rprime      DAStart - DAStart drvrPrime + w!
     Rctl        DAStart - DAStart drvrCtl + w!
     Rstatus DAStart - DAStart drvrStatus + w!
     Rclose      DAStart - DAStart drvrClose + w!
     Rflags      DAStart drvrFlags + w!
     Rdelay      DAStart drvrDelay + w!
     Remask      DAStart drvrEmask + w!
     RMenu       DAStart drvrMenu + w!
     Rname count dup  DAStart drvrName + c!
        DAStart drvrName + 1+ swap
        dup 31 > if drop 31 then cmove
     here DAstart -     DAStart DAlength + !
;

: DAlen DAlength + @ ;
\ get length word of external definition

\ **** DA glue macros

CODE DA.prelude
     LINK    A6,#-512 \ 512 bytes of local Forth stack
     MOVEM.L A0-A1,-(A7)   \ save registers
     MOVE.L  A6,A3     \ setup local loop return stack
     SUBA.L   #256,A3  \ in the low 256 local stack bytes
     MOVE.L  A0,-(A6) \ parameter block
     MOVE.L  A1,-(A6)      \ device control entry
     RTS \ just to indicate the MACHro stops here
END-CODE MACH

CODE DA.epilogue
     MOVE.L  (A6)+,D0 \ return code
     MOVEM.L (A7)+,A0-A1   \ restore registers
     UNLK    A6
     RTS
END-CODE MACH

CODE DA.Jiodone
     MOVE.L  (A6)+,D0 \ return code
     MOVEM.L (A7)+,A0-A1   \ restore registers
     UNLK    A6
     move.l  JIODone,A0
     movem.l d4-d7/a4-a6,-(a7)
     jsr     (a0)
     movem.l (a7)+,d4-d7/a4-a6
     RTS
END-CODE MACH

.TRAP _newptr,sys  $A51E

%0000000010100101 0 CONSTANT DAEmask

$1B4 CONSTANT SystemTask

\ ------------------------------------------
\ time manager and systemTask patch routine
\ this routine must reside in a block allocated
\ in the system heap through a pointer.
\ ------------------------------------------

header myTask 14 allot
6 CONSTANT taskPtr
HEADER myName
     DC.B    9,0,'Reminder'
header myTrap 4 allot
header myAlert 4 allot
header myString 256 allot

: alertMe
     MOVEM.L A0-A4/A6/D0-D7,-(A7)
     LINK    A6,#-128 \ 128 bytes of local Forth stack
     (call) frontwindow windowkind + @
     2 <> IF
        ['] myTrap @ SystemTask (call) SetTrapAddress
        ['] myString 0 0 0 (call) paramText
        ['] myAlert @ 0 (call) noteAlert drop
        ['] myAlert @ (call) freeAlert
        ['] myTask RmvTime
        ['] myTask (call) DisposPtr drop
     THEN
     UNLK    A6
     MOVEM.L (A7)+,A0-A4/A6/D0-D7
;

: wakeMe
     SystemTask (call) GetTrapAddr ['] myTrap !
     ['] alertMe SystemTask (call) SetTrapAddr
;

header mytask.end
' wakeme ' mytask - CONSTANT *wakeme \ task offset
' myAlert ' mytask - CONSTANT *myAlert \ alertID
' myString ' mytask - CONSTANT *myString \ alert string

\ ------------------------------------------
\ desk accessory code starts here.
\ ------------------------------------------

:DA reminder
     .ALIGN

( *** main desk accessory routines *** )
header myRes0 4 allot      \ local res ID=0 offset
header dlgText 256 allot

\ redefinition of cmove to make it
\ available locally

CODE cmove
     move.l  (a6)+,d0
     move.l  (a6)+,a1
     move.l  (a6)+,a0
     tst.l   d0
     ble.s   @2
@1   move.b  (a0)+,(a1)+
     subq.l  #1,d0
     bne.s   @1
@2   rts
END-CODE

\ ----------------
\ wakeup routine installation
\ ----------------

: install.wakeup
     ( delay alrtID msg | procHdl hSize taskBlock - )

     "proc ['] myRes0 @ (call) GetResource -> procHdl
     procHdl (call) getHandleSize -> hSize
        hSize MOVE.L (A6)+,D0
        _newPtr,sys
```

```
        MOVE.L   A0,-(A6) -> taskBlock
    procHdl @ taskBlock hSize cmove
    procHdl (call) releaseResource
      \ we have made a local copy of the wakeup routine
    taskBlock dup *wakeMe + swap taskPtr + !
        msg taskBlock *myString + 256 cmove
        alrtID taskBlock *myAlert + !
        alrtID (call) CouldAlert
    taskBlock InsTime
    taskBlock delay PrimeTime
        \ now the wakeup routine will wake up after
        \ the scheduled delay.
;

: getDrvrID ( dCtlEntry | - num )
    dCtlEntry dCtlRefNum + w@ l_ext
    1+ negate
;

: ownResID ( resID drvrID )
    5 shl + -16384 +
;
: Open ( parblk dce | DAWind Res0 - returncode )
    5 (call) sysbeep
        \ to get attention if automatically opened
    0 dce getDrvrID ownResID -> Res0
    dce dCtlWindow + @ -> DAWind
    DAWind 0= IF ( not open already )
        Res0 ['] myRes0 !
        Res0 0 -1 (call) getNewDialog -> DAWind
        DAWind  dce dCtlWindow + !
        \ store dialog pointer
        DAWind  dce dCtlRefNum + w@
                swap windowKind + w!
    ELSE
        DAWind (call) selectWindow
    THEN
    0
;
: Close ( parblk dce | - returncode )
    dce dCtlWindow +
    dup @ (call) DisposDialog
        0 swap ! ( so that Open will work again )
    0
;
: dialog-handler
    ( dlgPtr itemHit |
        itemType hItem rBox seconds - )

\ we get here if the OK button in the dialog
\ has been hit, therefore itemHit is always =1
\ - in our case. But it is nice to have itemHit
\ available, to be more general.
\ item #3 contains the appointment message
\ item #4 contains the delay in seconds
\ (decimal number string)

    dlgPtr 4 ^ itemType ^ hItem ^ rBox (call) GetDItem
    hItem ['] dlgText (call) GetIText
    ['] dlgText (call) StringToNum -> seconds
    seconds 0> IF
        dlgPtr 3 ^ itemType ^ hItem ^ rBox
                              (call) GetDItem
        hItem ['] dlgText (call) GetIText
        seconds 1000 w*
            ['] myres0 @ ['] dlgText install.wakeup
        ELSE 10 (call) sysbeep
        THEN
;

: Ctl ( parblk dce |
        DAWind event-rec dlgPtr itemHit - returncode )
    dce dCtlWindow + @ -> DAWind
    parblk csCode + w@ l_ext
    CASE
        accEvent OF
            2 DAWind windowKind + w!
                    \ set to dialog window
            parblk csParam + @ -> event-rec
            event-rec (call) IsDialogEvent
            IF  event-rec ^ dlgPtr ^ itemHit
                (call) Dialogselect
                IF dlgPtr itemHit dialog-handler THEN
            THEN
            DAWind  dce dCtlRefNum + w@
                swap windowKind + w!
                    \ reset windowkind
        ENDOF

    ENDCASE
    0
;
: DrOpen DA.Prelude Open DA.Epilogue ;
: DrClose DA.Prelude Close DA.Epilogue ;
: DrCtl DA.Prelude Ctl DA.JioDone ;
: DrStatus ;
: DrPrime ;

' DrOpen ' DrPrime ' DrCtl ' DrStatus ' DrClose
$7400 \ need lock, need time, need goodbye, ctl calls
60 DAEmask 0 \ delay mask menu
" Reminder" \ name
;DA


( write resource to file )
: $create-res ( str-addr - errcode )
    call CreateResFile
    call ResError L_ext
;
: $open-res ( addr | refNum - refNum or errcode )
    addr call OpenResFile -> refNum
    call ResError L_ext
    ?dup IF ELSE refNum THEN
;
: close-res ( refNum - errcode )
    call CloseResFile
    call ResError L_ext
;
: make-res ( addr len rtype ID name | - )
    addr len call PtrToHand
    abort" Could not create resource handle"
    rtype ID name call AddResource
;
: write-out ( filename | refnum - )
    filename $create-res
            abort" That resource file already exists"
    filename $open-res
    dup 0< abort" Open resource file failed"
    -> refnum
    refnum call UseResFile
    ['] reminder dup DALen
        "drvr 12 " Reminder" make-res
    ['] myTask ['] mytask.end over -
        "proc -16000 " wakeUp" make-res
        "proc -16000 call GetResource
        dup 80 call SetResAttrs
            ( 64: sysheap + 16: locked )
        call ChangedResource
    refnum close-res
            abort" Could not close resource file"
;
: make-DA
    " Reminder.rsrc" $delete drop
    " Reminder.rsrc" write-out
;
```

© The Definitive MacTutor, Vol. 4

Forth Forum

Direct 68881 Floating Point Access

Jörg Langowski
MacTutor Editorial Board
Grenoble, France
MacTutor Vol. 4 No. 4

Being the proud owner of a MacII since the beginning of this year, I was - like so many others - disappointed at the relative slowness of the SANE package. Even with the floating point coprocessor, the speedup is only a factor of 5-10 compared to the Mac SE. The coprocessor itself allows you to go much faster than that, and it is Apple's SANE implementation that slows down the operations.

The reason for this is, of course, that Apple is trying to guarantee the 'compatibility' of the SANE on the MacII with the old SANE implementation. That means that the results are supposed to be the same, down to the least significant bit (or even to the last guard bit?). The 68881 uses different algorithms for calculating transcendental functions than SANE does, therefore some of its built-in operations could not be used and had to be replaced by software. Of course, that slows down things a lot.

As a side note, I'm not at all in favor of such a strategy. Artificially restraining a high performance chip just because its results, although accurate enough, don't match the old - also accurate - results to the last bit seems a little exaggerated. Stable numerical algorithms should take into account the possibility that the hardware changes slightly and the machine errors are different, and they should be immune against such changes; or at least the new SANE should have had an option built in that uses the 68881 directly! *[Amen to that statement! -Ed]*

Many of the development systems for the Macintosh now come with the possibility to generate code that supports the floating point coprocessor directly, many others don't; working with Forth, we don't have a real problem since we can easily redefine our floating point operators. This month I'll show you how to do that.

The 68881

Let's first have a look at the floating point coprocessor itself.

The 68881 is accessed from the 68020 in a special address area. When the 68020 encounters an instruction of the form $Fxxx (previously the F line trap), it will set its function code lines (pins FC0-FC2) all to high, indicating 'CPU space'. It will then exchange information with the coprocessor's internal registers to perform the floating point operation requested.

This information exchange occurs automatically, it is part of the 68020's design; when using the floating point instructions, you don't notice the communication between the two processors at all. The 68881 appears as an extension to the 68020, just as if we had a new set of registers available, with special instructions operating on them.

The registers FP0-FP7 can each hold one extended precision (80-bit) floating point number, in the format given by the IEEE standard: bits 0-63 contain the mantissa, 64-78 the 15-bit exponent offset by 16383, and bit 79 the sign. The instruction set of the 68881 allows you to do floating point operations on any or between any two of these registers, and transfer data between them, the 68020's registers and memory.

Mach2's assembler fully supports all 68881 instructions. The redefinition of the words in the SANE vocabulary is therefore quite straightforward. Register D7 is used by Mach2 as the floating point stack pointer, all we have to do is to transfer floating point numbers from the FP stack to the 68881's registers, do the FP operation and transfer back the result.

The 68881 expects floating point numbers in a different format than SANE does. In order to keep FP numbers aligned to the boundaries of a 32-bit long word, which speeds up operations, an extended precision number *in memory* will be 96 bits long instead of 80; there is a 16 bit gap between the 64-bit mantissa and the 16-bit sign and exponent field. Since SANE and the floating point stack in Mach2 use 80 bit numbers, we have to convert their format when we use the coprocessor.

Listing 1 provides the macros **f>2**, **f>1** and **1>f** for this purpose. The general format for a binary floating point operation is then

```
f>2
fop.x   FPn, FPm
1>f
```

where fop.x is an extended format floating point instruction that operates on floating point registers FPn and FPm. The example always uses registers FP0 and FP1. Unary operations are encoded the same way, using only FP0.

In the example, I provided a new vocabulary **f68881** that contains redefinitions of the most important SANE operations for the 68881. Of course, the concept can be far extended. We have eight registers at our disposition and can use them to optimize more complicated numerical algorithms, using only assembly language. I might give some examples how to do this in a later column.

For the moment, let's content ourselves with the speed improvement that we have achieved so far, which is already remarkable. Some simple benchmarks are listed at the end of listing 1, and the results are given here:

```
( MacII, direct 68881 access, 100000 loops each )
bmark1 2  Secs 18  Ticks ok <0> [0]
bmark2 4  Secs 48  Ticks ok <0> [0]
bmark3 4  Secs 14  Ticks ok <0> [0]
bmark4 4  Secs 17  Ticks ok <0> [0]
bmark5 4  Secs 14  Ticks ok <0> [0]
( MacII, SANE w/68881, 10000 loops each )
smark1 15  Ticks ok <0> [0]
smark2 37  Secs 12  Ticks ok <0> [0]
smark3 24  Ticks ok <0> [0]
smark4 1  Secs 33  Ticks ok <0> [0]
smark5 1  Secs 33  Ticks ok <0> [0]
( Mac +, SANE, 10000 loops each )
 ok <0> [0]
```

© The Definitive MacTutor, Vol. 4

```
smark1 24  Ticks ok <0> [0]
smark2 181  Secs 54  Ticks ok <0> [0]
smark3 49  Ticks ok <0> [0]
smark4 4  Secs 47  Ticks ok <0> [0]
smark5 5  Secs 14  Ticks ok <0> [0]
```

For the Mac+, the **fnull1** and **fnull2** operations had been replaced by simple **fdrops**. As you see, the speedup going from Mac+ to MacII's SANE is not so breathtaking: a factor of 6 for the exponential, 4 for addition and subtraction; but when we access the 68881 directly, we gain another factor of 3 for simple addition and multiplication and 78 for the exponential. It is in the calculation of the transcendentals where the 68881 really shines.

Pop up menus

Someone approached me lately on the question of how to do pop up menu selection. Since the technical notes contain only sketchy references to popup menus at the time I write this (e.g. TN156, TN172), I'd like to give you a practical example how to use pop up menus from Mach2 in a simple way.

Listing 2 explains the process. The *PopUpMenuSelect* trap takes four parameters:

- a handle to the menu to be displayed (32 bits),
- the top and left global coordinate of the point at which to display the menu (2*16 bits),
- the menu item which should be positioned at that point for the default selection.

Although Mach2 knows the trap name, the interface to this routine is not (yet) correct, so we have to redefine it in assembler. Note that the point returned by the **@mouse** function is in local coordinates, while *PopUpMenuSelect* expects it in global coordinates.

The example defines a menu using the Mach2 interface; the menu is created with -1 as the insertion parameter (**-1 150 mymenu BOUNDS**) so that after insertion into the tasks's menu bar the menu will stay invisible (just as we did for the hierarchical menus). Note that a pop up menu has to be inserted into the menu bar before using it.

The content handler for the default Mach2 window is then rewritten so that on a mouse down event it will select the example pop up menu. The menu handler will just beep a number of times depending on the item selected. **dopop** activates the new content handler while **nopop** deactivates it.

Feedback dept.

This letter comes from Vassili Dzuba, Paris:

"In January's issue 'Mousehole Report', Alan Dall put in his wish list the ability to define 'ghost copies' of applications. Even without Unix' capability of defining links, this can be done with a small program using the *_Launch* trap. This program takes only 1K on the disk. The path name of the application to launch is stored in 'STR ' resource 1000. It's possible to set the creator and the bundle bit of the ghost to have it share the same icon as the real thing. Of course, double-clicking on a document can then launch the ghost instead of the application, but the slowing down is only marginal.

The program is the following

(using MPW's assembler):

```
  INCLUDE 'traps.a'
ghost MAIN
  MOVE.W #0,-(SP) ; the context data (_launch parameter)
  MOVE.L #'STR ',-(SP); 1st parameter of GetResource
  MOVE.W #1000,-(SP); 2nd parameter of GetResource
  _GetResource; handle to string in (SP)
  TST.L (SP); test if null handle (no resource available)
  BEQ.B exit; if null, go to exit
  MOVE.L (SP),A0
  MOVE.L (A0),(SP); handle dereferenced
  MOVE.L SP,A0; stack pointer in A0
  _Launch
exit _ExitToShell
  END
```

The resource file is something like this (using Rez format):

```
#include "types.r"
resource 'STR ' (1000) {
    "Sys:myDirectory:MyApp"
};
```

A ghost can be easily created using a small shell script (assuming the original ghost's directory being {MPW}dev) which sets up the string resource:

```
duplicate {MPW}'dev:ghost' {2}
echo '#include "types.r"'∂n∂
'resource '∂''STR '∂'' (1000) { "'{1}'" };'∂
| rez -a -o {2}
```

Assuming this script is named 'summon', the creation in the current directory of a ghost of MacPaint would be something like:

```
summon 'sys:appli ƒ:mac paint' 'macpaint.Ghost'
```

Sincerely yours"

Thank you, Vassili, for this helpful little utility.

Listing 1: direct access 68881 floating point words for Mach2

```
\ 68881 access, © J. Langowski/MacTutor Jan 1988
only forth also assembler also sane
vocabulary f68881
also f68881 definitions
code f>2
     add.l    #20,d7
     move.l   d7,a0
     move.l   -(a0),-(a7)    ; move mantissa
     move.l   -(a0),-(a7)    ; in two 32-bit chunks
     subq.l   #2,a7          ; 16-bit gap
     move.w   -(a0),-(a7)    ; move exponent + sign
     fmove.x  (a7)+,fp0      ; transfer from stack to fp0
     move.l   -(a0),-(a7)    ; same for fp1...
     move.l   -(a0),-(a7)
     subq.l   #2,a7
     move.w   -(a0),-(a7)
     fmove.x  (a7)+,fp1
     add.l    #10,a0  ; a0 points to 2nd fp stack item
     rts
end-code mach

code f>1
     add.l    #10,d7
     move.l   d7,a0
     move.l   -(a0),-(a7)
     move.l   -(a0),-(a7)
     subq.l   #2,a7
     move.w   -(a0),-(a7)
     fmove.x  (a7)+,fp0
     rts
```


© The Definitive MacTutor, Vol. 4

```
end-code mach

code 1>f
     fmove.x  fp0,-(a7)
     move.w   (a7)+,(a0)+
transfer exponent + sign
     addq.l   #2,a7  ; skip16
bit gap
     move.l   (a7)+,(a0)+
transfer mantissa
     move.l   (a7)+,(a0)+ 2
steps
     sub.l    #10,d7 ; adjust
FP stack pointer
     rts
end-code mach

( note: f>1 or f>2 and 1>f
should always occur in pairs
since 1>f expects A0 to point
to second floating point
stack position; this is
assured by f>1 and f>2 )

code f+
     f>2
     fadd.x   fp1,fp0
     1>f
     rts
end-code

code f-
     f>2
     fsub.x   fp1,fp0
     1>f
     rts
end-code

code f/
     f>2
     fdiv.x   fp1,fp0
     1>f
     rts
end-code

code f*
     f>2
     fmul.x   fp1,fp0
     1>f
     rts
end-code

code fmod
     f>2
     fmod.x   fp1,fp0
     1>f
     rts
end-code

code frem
     f>2
     frem.x   fp1,fp0
     1>f
     rts
end-code

code fabs
     f>1
     fabs.x   fp0
     1>f
     rts
end-code

code facos
     f>1
     facos.x  fp0
     1>f
     rts
end-code

code fasin
     f>1
     fasin.x  fp0
     1>f
     rts
end-code

code fatan
     f>1
     fatan.x  fp0
     1>f
     rts
end-code

code fatanh
     f>1
     fatanh.x fp0
     1>f
     rts
end-code

code fcos
     f>1
     fcos.x   fp0
     1>f
     rts
end-code

code fcosh
     f>1
     fcosh.x  fp0
     1>f
     rts
end-code

code fe^x
     f>1
     fetox.x  fp0
     1>f
     rts
end-code

code fe^x-1
     f>1
     fetoxm1.x fp0
     1>f
     rts
end-code

code fgetexp
     f>1
     fgetexp.x fp0
     1>f
     rts
end-code

code fgetman
     f>1
     fgetexp.x fp0
     1>f
     rts
end-code

code fint
     f>1
     fint.x   fp0
     1>f
     rts
end-code

code fintrz
     f>1
     fintrz.x fp0
     1>f
     rts
end-code

code fln
     f>1
     flogn.x  fp0
     1>f
     rts
end-code

code fln+1
     f>1
     flognp1.x fp0
     1>f
     rts
end-code

code flog10
     f>1
     flog10.x fp0
     1>f
     rts
end-code

code flog2
     f>1
     flog2.x  fp0
     1>f
     rts
end-code

code fneg
     f>1
     fneg.x   fp0
     1>f
     rts
end-code

code fsin
     f>1
     fsin.x   fp0
     1>f
     rts
end-code

code fsinh
     f>1
     fsinh.x  fp0
     1>f
     rts
end-code

code fsqrt
     f>1
     fsqrt.x  fp0
     1>f
     rts
end-code

code ftan
     f>1
     ftan.x   fp0
     1>f
     rts
end-code

code ftanh
     f>1
     ftanh.x  fp0
     1>f
     rts
end-code

code f10^x
     f>1
     ftentox.x fp0
     1>f
     rts
end-code

code f2^x
     f>1
     ftwotox.x fp0
     1>f
     rts
end-code

fp

also forth definitions
code fnull1
     f>1
     1>f
     rts
end-code

code fnull2
     f>2
     1>f
     rts
end-code

: bmark1 counter 1.0
     100000 0 do fdup fnull1
fdrop loop
timer fdrop ;

: bmark2 counter 4.3352
     100000 0 do fdup fe^x
fdrop loop
timer fdrop ;

: bmark3 counter
     3.5 4.5 100000 0 do
fover fover fnull2 fdrop loop
timer fdrop fdrop ;

: bmark4 counter
     3.5 4.5 100000 0 do
fover fover f+ fdrop loop
timer fdrop fdrop ;

: bmark5 counter
     3.5 4.5 100000 0 do
fover fover f* fdrop loop
timer fdrop fdrop ;

also sane

: smark1 counter
     1.0 10000 0 do fdup
fnull1 fdrop loop
timer fdrop ;

: smark2 counter
     4.3352 10000 0 do fdup
fe^x fdrop loop
timer fdrop ;

: smark3 counter
     3.5 4.5 10000 0 do fover
fover fnull2 fdrop loop
```

```
timer fdrop fdrop ;

: smark4 counter
    3.5 4.5 10000 0 do fover fover f+ fdrop loop
timer fdrop fdrop ;

: smark5 counter
    3.5 4.5 10000 0 do fover fover f* fdrop loop
timer fdrop fdrop ;
```

Listing 2: Pop up menus

```
\ popup menu interface from Forth
\ © J. Langowski/MacTutor Jan 1988

\ FUNCTION PopUpMenuSelect
\ (menu: MenuHandle; top, left, popUpItem: INTEGER):
\            LONGINT;
\    INLINE   $A80B;

code popup ( hMenu top left item# | menuID item  - )
     exg      d4,a7
     clr.l    -(a7)
     move.l   12(a6),-(a7)   \ handle
     move.w   10(a6),-(a7)   \ top
     move.w   6(a6),-(a7)    \ left
     move.w   2(a6),-(a7)    \ item
     adda.l   #16,A6
     _popupmenuselect
     clr.l    d0
     clr.l    d1
     move.w   (a7)+,d0
     move.w   (a7)+,d1
     exg      d4,a7
     move.l   d0,-(a6)
     move.l   d1,-(a6)
     rts
end-code

CODE  @MOUSE
     SUBQ.L    #4,A6
     MOVE.L    A6,-(A7)
     _GETMOUSE
     RTS
END-CODE

CODE unpack
     MOVE.L (A6),D0
     CLR.L    D1
     MOVE.W D0,D1
     CLR.W  D0
     SWAP.W D0
     MOVE.L D0,(A6)
     MOVE.L D1,-(A6)
     RTS
END-CODE MACH

108 user taskmenubar
152 user content-hook

NEW.MENU myMenu
" TestMenu" myMenu TITLE
-1 150 myMenu BOUNDS \ invisible menu
" Item 1;Item 2;Item 3" myMenu ITEMS

taskmenubar @ mymenu add
call drawmenubar

: beep
     ?dup if
               0 do 5 call sysbeep loop
     then
;

: content-handler ( | pt - )
     @mouse -> pt
     ^ pt call localtoglobal
     mymenu @
     pt unpack
     1
               popup
     beep drop
     run-content
;

: dopop ['] content-handler content-hook ! ;
: nopop ['] run-content content-hook ! ;
```

French Cracker Scandal - the Sequel

Many of you might have wondered what has come out of the story we published in the August 1987 issue, about Faraglace, the guy who deprotected 4th Dimension and was supposed to pay the rest of his life for his 'youthful sin'.

The Apple Expo in Paris saw the situation evolve. In a discussion with Laurent Ribardiere, the author of Fourth Dimension, it appeared that people at ACI - including himself - had really thought that they were on the track of an international pirate network. When, in fact, only three people were involved making business out of this and that the fourth guy - Faraglace - had met them by chance, not knowing what they were up to, was not clear to ACI. It was for that reason that the four were initially judged together.

After the discussion with Laurent Ribardiere and Marylene Delbourg-Delphis (ACI's president), ACI is now willing to separate the two cases - the cracker and the pirates - for the appeal. This could help clear up the issues and give Faraglace the chance to have a fairer judgment. We don't know the position of the two other parties, Microsoft and Apple. Apple is apparently willing to follow ACI's lead.

The Faraglace case has also served as a trigger for something more general. It has appeared to quite a few members of the Mac developer/user community in France that the relation between software authors, publishers and end users is still characterized by profound mutual distrust in too many cases. Developers and publishers assume that users pirate back and forth, and the user often has lost all hopes to get a reasonable service from the publisher, as far as upgrades, bug-free products and fulfillment of expectations raised by advertising hype goes.

The Faraglace case seemed to have been aggravated by this atmosphere of mutual distrust. This was heavily discussed on the Calvacom bulletin board, and the general feeling was that one should have some means of organized action to change the situation into one of mutual respect between the parties involved in software evolution. The basic idea had been stated already some months earlier by Philippe Chatiliez, rock musician and sysop of the MIDI section on the bulletin board:

"Sure, software is intellectual property. But whatever is created by a user of the software, be it a written document, a drawing, music, another program or even a game score, is also intellectual property. Therefore the publisher has a responsibility towards the user to protect his/her rights just as the user has the responsibility to respect the author's rights. This creates a soli-

darity between the parties."

It was from this idea that *Solidarsoft* was founded. It is a non-profit association of software developers, publishers and users which aims at improving the flow of information between these groups. Solidarsoft wants to promote the idea of mutual respect, trying to cut down on piratage and things like abusive licensing conditions or copy protection at the same time.

First activities of Solidarsoft include creating a 'quality label' for software products and distributors, and acting as a shareware clearinghouse, helping for instance to overcome administrative difficulties that come up when a company uses a shareware product and needs an invoice.

If you want to know more about the activities of Solidarsoft, write to:

Solidarsoft
66 boulevard Exelmans
75016 Paris, France.

Major bulletin board addresses:
Compuserve: 71260,660. 73200,2025.
Pan: chat.
BIX: chatchat.
The Source: BEL010.
Calvacom: Soft10.

Forth Forum
Formatted Text with Style

Jörg Langowski
MacTutor Editorial Board
MacTutor Vol. 4 No. 5

TextEdit with Style(s)

TextEdit has always been one of the major conveniences of the Macintosh. Life (as far as programming is concerned) is so much simpler with a standard editing package that takes care of line break, word wrap, text insertion, justification etc. for you, instead of having to write it yourself.

One major reason why TextEdit was inadequate for e.g. word processing had always been that the text style was fixed for one text edit record, i.e. for one piece of edited text. Changing fonts, boldfacing, or changing the size of only one word or one phrase in the text was not possible.

The new implementation of TextEdit that is described in IM Vol.V finally introduces this possibility. TextEdit now recognizes two types of TE records: the old one, having the same text style all over the text, and a new format where *style information* is associated and displayed with the text. Although contained in IM V, the new TextEdit traps are available (System 4.2 and later) on the Mac Plus as well as on the SE and MacII.

A second inconvenience has been resolved: When working on new type TE records, TECopy and TEPaste now uses the *desk scrap* (i.e. the general clipboard) for cutting/pasting directly. It is no more necessary to convert between the internal TE scrap and the desk scrap on activate/deactivate events. For old TE records, TECopy and TEPaste still only use the internal scrap (for compatibility).

The new text edit is not yet used very widely, but it is worth it. Therefore I thought it might be a good idea to show its function on a practical example in Mach2 Forth. It is based on the 'skeleton editor' on the Mach2 demo disk, but contains a lot of enhancements, such as horizontal scrolling and scroll thumbs that work, and of course the text style support.

The new TE record

How does the system distinguish between the new and the old type TE record?

Mostly, the structure of the new record is the same as the old one, with a couple of exceptions: The txSize field of a new TE record contains -1 (word, offset $50 from beginning of record). In that case, the fields txFont and txFace (offset $4A and $4C) are combined to hold the handle of what is called a *style record*, a structure that contains the style information associated with the text. Of course, the information previously contained in the *txFont, txFace and txSize* fields - font, text face, and font size - is not meaningful anymore, since it may change over the length of the text.

Furthermore, the fields *lineHeight* and *fontAscent* (offset $18 and $1A) may now also contain -1, in which case the line height and ascent will be calculated individually for each line; if they contain positive numbers, the TErecord has fixed line height and ascent as before.

The style record

This structure contains the style information that belongs to the TErecord. Its layout is give in IM V-261 and I'll repeat it here (Pascal style):

```
TYPE
TEStyleHandle = ^TEStylePtr;
TEStylePtr = ^TEStyleRec;
TEStyleRec = RECORD
             nRuns:       INTEGER;
             nStyles:     INTEGER;
             styleTab:    STHandle;
             lhTab:       LHHandle;
             teRefCon:    LONGINT;
             nullStyle:   nullSTHandle;
             runs:        ARRAY [0..0] OF StyleRun
         END;

StyleRun = RECORD
             startChar:   INTEGER;
             styleIndex:  INTEGER
         END;
```

The text is characterized by a succession of *style runs*, characters of the same style in the TE record. The system knows where they start (*startChar*) and the index *styleIndex* of the corresponding style in a *style table*. Each style record contains information about how many style runs and how many different styles there are in the text (*nruns, nStyles*), and handles to the style and line height tables (*styleTab, lhTab*). *teRefCon* can be used freely by the application, and *nullStyle* contains a handle to the style table that is used when the selection is empty.

The style table contains an entry for each different style encountered in the text:

```
TYPE
STHandle = ^STPtr;
StPtr = ^TEStyleTable;
TEStyleTable = ARRAY [0..0] OF STElement;

STElement = RECORD
              stCount:     INTEGER;
              stHeight:    INTEGER;
              stAscent:    INTEGER;
              stFont:      INTEGER;
              stFace:      Style;
              stSize:      INTEGER;
```


© The Definitive MacTutor, Vol. 4

```
        stColor:    RGBColor;
    END;
```

stCount indicates how often this style occurs in the text; the remaining fields indicate the text style.

Since the line height now may differ from line to line, there is another table, the *line-height table*, which contains an entry for each line in the *lineStarts* array of the TE record.

```
TYPE
LHHandle = ^LHPtr;
LHPtr = ^LHTable;
LHTable = ARRAY [0..0] OF LHElement;

LHElement = RECORD
            lhHeight:   INTEGER;
            lhAscent:   INTEGER;
        END;
```

For each line, *lhHeight* and *lhAscent* contain the maximum height and ascent present in that line.

Fortunately, you don't have to remember any of this information when using the new Text Edit. For straightforward text editing, all we need to know is that the TECopy, TECut, TEKey and TEDelete routines work as before, but Copy and Cut store a second type of scrap, 'styl', in the desk scrap. When you do a TEPaste, this information is not used and only the text is pasted into the edit record using the style that was present at the insertion point. For pasting 'with style', a new routine exists, *TEStylPaste*. This is one of 10 routines that are called through the new trap *TEDispatch*. This trap will look at a selector (integer) on top of stack and call the new text edit routines accordingly.

Mach2 implements the new TE routines using their names and compiling the glue code for the dispatch routine automatically. One routine, *TEStylInsert*, has been left out accidentally, and is redefined in the example listing. Two more new traps exist: *TEStylNew* and *TEGetOffset*. Mach2 spells the former 'TEStyleNew', so watch out for that.

For simple use as in the example, the most important new routines that you have to remember are *TEStylNew*, *TEStylPaste*, *TEGetOffset*, *TEGetStyle*, and *TESetStyle*.

TEStylNew is just like *TENew*, except that it creates a new type TE record. *TEStylPaste* will use the 'styl' information in the scrap, if present, for the text to be pasted.

TEGetOffset works with both old and new type TE records and finds the character offset in the text corresponding to a certain point in the window (local coordinates). This is necessary for updating the font, size and style menu information after a change of the insertion point by a mouse click.

TEGetStyle and *TESetStyle*are used to get and set style information at the selection of insertion point in the text. They work with another data structure, the *TextStyle:*

```
TYPE TextStyle = RECORD
            tsFont:  INTEGER;
            tsFace:  Style;
            tsSize:  INTEGER;
            tsColor: RGBColor
        END;
```

This record contains the text style information that is passed to/from the TE record.

All these routines are, of course, documented in IM Vol. V. Examples on how to call them can be found in the program listing. Feel free to experiment with the other new TE routines.

The example will create an editor window with vertical and horizontal scroll bars and read the first 1024 characters from a text file. You can use this text for playing around with different style settings (using the menus I provided). Doing this, you will notice that the **Style** menu doesn't seem to have any effect on the text. Fonts and font sizes may be changed, but underlining, boldfacing etc. seems to be impossible. However, when you set those text attributes for a certain selection and then click at different positions in the text, you will see that the Style menu changes appropriately, checkmarks appear and disappear in the right positions. Therefore I believe that the style changes are well recorded in the style record/style table; only the text display doesn't seem to function. Is this a bug still present in the new TextEdit, or have I overlooked something obvious (question to you, the readers)?

You see that editing multi-style text is not too difficult with the new TE routines, since they take care of almost everything. For saving and loading files, you would have to provide code that reads/writes the style record information together with the text, this also seems pretty straightforward. Printing a new style text edit record is a completely different ballgame, since no printing support exists (yet?) that is quite as extensive as the new TextEdit. You would have to extract the style information for each line, using the style record, style and line height tables, and write the printing code from scratch. More difficult, but probably a good 'exercise for the reader'.

Bugs in the SCSI driver?

Some remarks have been received that the SCSI code published in V4#1/2 was buggy, and someone seems to have crashed his hard disk using it. I feel sorry if application of that example (which was however marked 'experimental') should have caused trouble to anyone. The only excuse I have is that it was said that the code had been *shown* to work only for two types of disk, the Q280 and the ST225N, the only ones available to me for testing at that time.

Let me point out here that we can only do our best to give examples of Macintosh programming, but never guarantee that everything works 100% and bug-free. Feel free to publish your comments, preferably with corrections, in MacTutor, that's what we're here for. But for any example that is printed here or published on the source code disks, be prepared that something might go wrong, and especially DON'T run it from the disk that has your only copy of that 40-page report on it that is due in two days! Play it safe and use floppies that have nothing valuable on them. I learned my lesson when I crashed my own hard disk while writing the Forth example for the February article.

I wish you good luck with this month's example. The source

code disk contains two compiled versions, one for Mac Plus/SE/II, and one for Mac II only (contains 68020 code). As far as I have checked them, they don't crash and work under MultiFinder. But you never know...

Till next month, happy threading.

Listing 1: Text Edit Example with Style

```
\ New Text Edit example
\ J. Langowski for Mac Tutor March 1988
\ derived from
\ Editor Shell Example Program on Mach 2 demo disk

\ found two 'features' of the new text edit while
\ experimenting:
\ a. when the insertion point is at a boundary between
\    two different styles, the text typed will be TEKeyed
\    according to the style BEFORE the insertion point,
\    while TEGetStyle will return style information from
\    AFTER the insertion point.
\ b. Although the text face seems to be set inside the style \
record and properly associated with the text
\    (TEGetStyle returns the correct information after the
\    text face has been changed), the text is always drawn
\    plain text style. The font and size changes work OK.
\

only forth definitions
also assembler also mac

\ ***** constants

300 CONSTANT APPLEID
310 CONSTANT FILEID
320 CONSTANT EDITID
330 CONSTANT SIZEID
340 CONSTANT FontID
350 CONSTANT StyleID

20  CONSTANT InUpArrow
21  CONSTANT InDownArrow
22  CONSTANT InPageUp
23  CONSTANT InPageDown
129 CONSTANT InThumb

$44525652 Constant 'drvr
$464F4E54 Constant 'font

%00000100000000000 CONSTANT ShiftMask

$10 CONSTANT portRect \ Grafport rectangle
$6E CONSTANT wVisible \ visible flag [byte]

\ text edit equates
0  CONSTANT  teDestRect
8  CONSTANT  teViewRect
$C  CONSTANT selRect
$18 CONSTANT teLineHite
$1A CONSTANT teFontAscent
$1C CONSTANT teSelPoint
$20 CONSTANT teSelStart
$22 CONSTANT teSelEnd
$38 CONSTANT teCarOn
$39 CONSTANT teCarAct
$3C CONSTANT teLength
$3E CONSTANT teTextH
$48 CONSTANT teCROnly
$4A CONSTANT teFont
$4C CONSTANT teFace
$4A CONSTANT teStylHandle
      \ handle to style record for new
      \ text edit. Never accessed directly
$4E CONSTANT teMode \ text mode [word]
$50 CONSTANT teSize \ font size [word]
   \ teFont, teFace, teMode, teSize are only
   \ meaningful for old style TE records.
   \ for a new style record, teSize contains -1.
   \ in that case, teFont and teFace together contain
   \ the handle to the style record.
$5E CONSTANT teNLines
$60 CONSTANT    teLines

$AB0 CONSTANT TEScrpLength
$AB4 CONSTANT TEScrpHandle

\ Event Record Equates
$0 CONSTANT  What      ( event code [word] )
$2 CONSTANT  Message   ( event message [long] )
$6 CONSTANT  When      ( ticks since start-up  [long] )
$A CONSTANT  Where     ( mouse loc. pt. global [long] )
$E CONSTANT  Modifiers ( modifier flags  [word] )

$0A CONSTANT LF     ( ascii 'linefeed' )
$20 CONSTANT SP     ( ascii 'space' )

create applestring 01 C, $14 C, \ Apple symbol

\ ***** variables

VARIABLE TEHandle      ( handle for text edit record )
VARIABLE TERect  4 VALLOT
          ( Text Edit view rectangle )
VARIABLE SIZE    ( item# of current textsize )
VARIABLE DESKNAME 252 VALLOT  \ holds name of desk accessory
VARIABLE FONTNAME 252 VALLOT  \ holds font name selected
VARIABLE ITEMNAME 60 VALLOT
   \ receives menu item name
VARIABLE MyStyle 8 VALLOT
   \ text style record for our private use
   \ fields of MyStyle:
      0 CONSTANT tsFont
      2 CONSTANT tsFace
      4 CONSTANT tsSize
      6 CONSTANT RGBColor
VARIABLE currentFont \ menu ID of font in use
VARIABLE #fonts     \ # of currently installed fonts
VARIABLE currentSize  \ menu ID of size in use

76  USER AbortHook
152 USER ContentHook
160 USER GrowHook
164 USER CloseBoxHook
168 USER UpdateHook
172 USER ActivateHook
202 USER CAction \ control action routine vector

\ ***** glue routines for new text edit
\
\ TEStylNew ( destRect viewRect - TEHandle )
\ is misspelled 'TEStyleNew' in the Mach2 trap definitions,
\ but implemented. So are TEGetOffset and most of the
\ other new text edit routines that are called through
\ TEDispatch.
\ One exception is TEStylInsert, which we are defining
\ here:

CODE TEStylInsert ( text length hST hTE - )
   EXG D4,A7
   MOVE.L $C(A6),-(A7) \ pointer to text
   MOVE.L $8(A6),-(A7) \ length of text
   MOVE.L $4(A6),-(A7) \ style record handle
   MOVE.L (A6),-(A7)   \ TE record handle
   ADDA.W #$10,A6
   MOVE.W #$7,-(A7)
```


© The Definitive MacTutor, Vol. 4

```
   _TEDispatch
   EXG D4,A7
   RTS
END-CODE

\ ***** windows, menus, controls, tasks etc. *****

NEW.WINDOW Editor
" Editor" Editor TITLE
42 4 330 507 Editor BOUNDS
DOCUMENT INVISIBLE CLOSEBOX GROWBOX
   Editor ITEMS

200 1000 TERMINAL EditTask

NEW.MBAR EditBar

NEW.MENU AppleMenu
APPLESTRING AppleMenu TITLE
0 APPLEID AppleMenu BOUNDS
" About Editor ...;(-" AppleMenu ITEMS

NEW.MENU FileMenu
" File" FileMenu TITLE
0 FileID FileMenu BOUNDS
" New/N;Open.../O;Close;Save;Save as...;Revert to
Original;(Print"
   FileMenu ITEMS

NEW.MENU EditMenu
" Edit" EditMenu TITLE
0 EDITID EditMenu BOUNDS
" (Undo/Z;(-;Cut/K;Copy/C;Paste/V;Clear"
   EditMenu ITEMS

NEW.MENU FontMenu
" Font" FontMenu TITLE
0 FontID FontMenu BOUNDS
"  (Fonts<I;(-" FontMenu ITEMS

NEW.MENU SizeMenu
" Size" SizeMenu TITLE
0 SizeID SizeMenu BOUNDS
"  9 Point; 10 Point; 12 Point; 14 Point; 18 Point; 20 Point;
24 Point"
   SizeMenu ITEMS

CREATE SizeIDTable
0 , 0 , 0 c, \ no menu IDs for sizes 0 thru 8
1 c, 2 c, 0 c, 3 c, \  9,10,-,12
0 c, 4 c, 0 c, 0 c, \ -,14,-,-
0 c, 5 c, 0 c, 6 c, \ -,18,-,20
0 c, 0 c, 0 c, 7 c, \ -,-,-,24

CREATE SizeTable
0 c, 9 c, 10 c, 12 c, 14 c, 18 c, 20 c, 24 c,


NEW.MENU StyleMenu
" Style" StyleMenu TITLE
0 StyleID StyleMenu BOUNDS
"  Plain/P; Bold/B<B; Italic/I<I; Underline/U<U; Outline<O;
Shadow<S; Condense; Extend"
   StyleMenu ITEMS

NEW.CONTROL Scroll
VSCROLLBAR VISIBLE 100 0 Scroll ITEMS
VARIABLE lastVs
400 CONSTANT maxVs

NEW.CONTROL hScroll
HSCROLLBAR VISIBLE 100 0 hScroll ITEMS
VARIABLE lastHs

100 CONSTANT maxHs

: CHECK  ( menuhandle item# flag - )  ( checking a menu item )
   CALL CheckItem ;

: =string ( aStr bStr | - flag )
   aStr count 65536 * bStr count rot + swap
   call CmpString 0=
;

CODE @TEHandle
   MOVE.L TEHandle,-(A6)      RTS
END-CODE

: Shift?   (   -  f )  \ checks the event record to see if the
      \ shift key was pressed.
   EVENT-RECORD Modifiers + W@ ShiftMask AND
   IF -1 ELSE 0 THEN
;


: adjustFontMenu
\ adjust font menu and currentFont variable
   myStyle w@ ( font ID ) fontName call getFname
   #fonts @ 0 DO
      fontMenu @ i itemName call GetItem
      itemName fontName =string
      IF  FontMenu @ currentFont @ 0 check
         \ uncheck previous font selection
          FontMenu @ i -1 check
          i currentFont !
          leave
      THEN
   LOOP
;

: adjustStyleMenu ( | face - )
   myStyle tsFace + w@ -> face
   8 0 DO
     1 i scale face and ( get style bit )
     if -1 else 0 then
     styleMenu @ i 2+ rot check
   LOOP
;

: adjustSizeMenu
   SizeMenu @ currentSize @ 0 check
   myStyle tsSize + w@ ( size )
   SizeIDTable + c@ ( sizeID )
   dup currentSize !
   SizeMenu @ swap -1 check
;

: getCurrentStyle ( | LHite FAsc - )
   ( updates variable currentFont )
   ( size and Face kept in myStyle )
   ( LHite and FAsc are currently not used )

   @TEHandle @ teselStart + w@
      \ get start of selection (or insertion point)
   ( offset ) myStyle ^ LHite ^ FAsc @TEHandle
      call TEGetStyle

   adjustFontMenu
   adjustStyleMenu
   adjustSizeMenu
;

: AdjustTERect
   \ adjust terect size for the presence of scrollbars
   portRect Editor + 4 + W@   ( get bottom coord )
   16 -   ( subtract 16 for height of scrollbar )
   teViewRect @TEHandle @ + 4 + W!    \ store new coord back
```

```
in text edit record

   portRect Editor + 6 + W@   ( get right coord )
   16 -   ( subtract 16 for width of scrollbar )
   teViewRect @TEHandle @ + 6 + W!
;

: Visible? (   -  f   ) \ checks visible flag in window record
   Editor wVisible + C@
;

\ ***** event handlers *****

: ACTIVATE-HANDLER
   RUN-ACTIVATE
   EVENT-RECORD Modifiers + W@   ( get modifiers word )
   1 AND IF
      @TEHandle CALL TEActivate
      getCurrentStyle
   ELSE
      @TEHandle CALL TEDeactivate
   THEN
;


: UPDATE-HANDLER
   Editor CALL SetPort
   AdjustTERect
   Editor CALL BeginUpdate
      Editor CALL DrawControls
      Editor CALL DrawGrowIcon
      Editor portRect + @TEHandle CALL TEUpdate
   Editor CALL EndUpdate
;

: CONTENT-HANDLER ( | theMouse - )
   RUN-CONTENT
   Editor CALL SetPort

   ^ theMouse CALL GetMouse
   theMouse @TEHandle @ TEViewRect +
      call PtInRect
   IF
   theMouse Shift? @TEHandle CALL TEClick
   getCurrentStyle
   THEN
;

: CLOSEBOX-HANDLER
   Editor \ windowpointer
   EVENT-RECORD Where + @ call TrackGoAwayIF Editor CALL
HideWindow THEN
;


\ **** main editor example code *****

: POP-UP
   Editor CALL ShowWindow
   Editor CALL SelectWindow
   EditBar @ CALL SetMenuBar
   CALL DrawMenuBar
;

: ShutDown
   Editor CALL HideWindow
   PAUSE     ( PAUSE so that the i/o task can
         have a turn and handle the
         deactivate event generated by
          the closing of the window )
   MACH.MBAR ( MACH menubar back on screen )
   @TEHandle CALL TEDispose
;

: SetScrollLimits
   Scroll @ Ø CALL SetMinCtl
   Scroll @ maxVs CALL SetMaxCtl
   Scroll @ Ø CALL SetCtlValue
   Ø lastVs !
   hScroll @ Ø CALL SetMinCtl
   hScroll @ maxHs CALL SetMaxCtl
   hScroll @ Ø CALL SetCtlValue
   Ø lastHs !
;


: EditFile   (   | char exitflag -   )
   BEGIN
      Visible?
      IF
      ?TERMINAL IF
         KEY -> char   ( get the character )
         char 14 = IF
            Ø -> exitflag ( if cmd '.' exit )
         ELSE
            char @TEHandle CALL TEKey
         ( else insert )
            1 -> exitflag       ( char )
         THEN
      ELSE
         1 -> exitflag
         ( if no key pressed, keep looping )
      THEN

      ELSE
      Ø -> exitflag
      ( if window's been closed, exit )
      THEN
   exitflag  ( check exit condition )
   WHILE
      @TEHandle CALL TEIdle
   REPEAT
;


: Open
   Pop-Up
   Editor CALL SetPort

   TERect TERect CALL TEStyleNew TEHandle !
      \ get new style TE record
   -1 teCROnly @TEHANDLE @ + W!       ( no word wrap )
   -1 teCarAct @TEHandle @ + C!       ( activate caret )
   -1 @TEHandle call TEAutoView
      \ enable auto scroll

   ( get the first 1K of text )
   Ø VIRTUAL 1024 Ø @TEHandle TEStylinsert
   Ø Ø @TEHandle CALL TESetSelect
   15 ( doAll ) myStyle -1 ( redraw) @TEHandle
      call TESetStyle
   adjustFontMenu
   adjustStyleMenu
   adjustSizeMenu

   AdjustTERect ( initialize the text )
   PortRect Editor + @TEHandle CALL TEUpdate
   @TEHandle CALL TEDeactivate
   @TEHandle CALL TEActivate

   SetScrollLimits
   Editor CALL DrawControls      Editor CALL DrawGrowIcon

   ['] UPDATE-HANDLER UpdateHook !
   ['] CONTENT-HANDLER ContentHook !
   ['] ACTIVATE-HANDLER ActivateHook !
```

```
   ['] CLOSEBOX-HANDLER CloseBoxHook !
;


\ ***** menu handlers *****

: HandleDeskAcc ( item# - )
   APPLEMENU @ SWAP DESKNAME CALL GETITEM
   DESKNAME CALL OPENDESKACC
   DROP
;

: DO-APPLE ( item# - )
   dup 1 = IF
       ( AboutEdit ) ( About Editor ... )
       drop
   ELSE HandleDeskAcc
   THEN
;

: NewFile ;
: OpenFile ;
: CloseFile ;
: SaveFile ;
: SaveAs ;
: Revert ;

: DO-FILE ( item# - )
   CASE
   1 OF   NewFile      ENDOF
   2 OF   OpenFile     ENDOF
   3 OF   CloseFile    ENDOF
   4 OF   SaveFile     ENDOF
   5 OF   SaveAs       ENDOF
   6 OF   Revert       ENDOF
   ENDCASE
;

: DO-EDIT ( item# - )
   CASE
   1 OF   ( TEUndo )              ENDOF
   3 OF   @TEHandle CALL TECutENDOF
   4 OF   @TEHandle CALL TECopy   ENDOF
   5 OF   @TEHandle CALL TEStylPaste  ENDOF
   6 OF   @TEHandle CALL TEDelete ENDOF
   ENDCASE
;

: DO-Font ( item# | fontID - )
   FontMenu @ item# Fontname call getitem
   Fontname ^ fontID call getFNum
   ^ fontID w@ myStyle w!
      \ put into tsFont field of style record
   1 ( doFont) myStyle -1 ( redraw) @TEHandle
      call TESetStyle
   FontMenu @ currentFont @ 0 check
   FontMenu @ item# -1 check
   item# currentFont !
;

: Do-Style ( item# | facefield - )
   myStyle tsFace + -> facefield
   item# CASE
      1 OF ( plain text )
        0 facefield w!
      ENDOF

      facefield w@
      1 item# 2- scale xor
      facefield w! \ flip bit
   ENDCASE

   2 ( doFace) myStyle -1 ( redraw) @TEHandle
      call TESetStyle
   adjustStyleMenu
;


: Do-Size  ( item# - )
   SizeTable + c@
   myStyle tsSize + w!
   4 ( doSize) myStyle -1 ( redraw) @TEHandle
      call TESetStyle
   adjustSizeMenu
;

: MBAR-HANDLER  ( item# menuID - )
   CASE
   APPLEID OF DO-APPLE ENDOF
   FILEID OF DO-FILE   ENDOF
   EDITID OF DO-EDIT          ENDOF
   FontID OF DO-Font   ENDOF
   SIZEID OF DO-Size          ENDOF
   STYLEID   OF DO-Style      ENDOF
   ENDCASE
   0 CALL HILITEMENU
;


\ ***** control action routines *****

\ A control action routine specifies what action should take
\ place WHILE a control is being held down.

: ScrollText  ( dv  dh  -  )
   dh dv @TEHandle CALL TEScroll
;

: DO-Scroll ( part-code  | ctlvalue  -  )
   part-code
   CASE
      inuparrow OF  Scroll @ CALL GetCtlValue  -> ctlvalue
      ctlvalue 0= NOT
          IF
         Scroll @ ctlvalue 1- call SetCtlValue
         5 0  ScrollText
          THEN
      ENDOF

      indownarrow OF  Scroll @ call GetCtlValue  -> ctlvalue
      ctlvalue maxVs = NOT
          IF
         Scroll @ ctlvalue 1+ call SetCtlValue
         -5 0 ScrollText
          THEN
      ENDOF

      inpageup  OF  Scroll @ call GetCtlValue  -> ctlvalue
      ctlvalue 0= NOT
          IF
         Scroll @ ctlvalue 5 - call SetCtlValue
         25 0  ScrollText
          THEN
      ENDOF

      inpagedown OF  Scroll @ call GetCtlValue   -> ctlvalue
      ctlvalue maxVs = NOT
           IF
         Scroll @ ctlvalue 5 + call SetCtlValue
         -25 0  ScrollText
          THEN
      ENDOF
   ENDCASE
   Scroll @ call GetCtlValue lastVs !
;
```

```
: DO-hScroll ( part-code  | ctlvalue  -  )
   part-code
   CASE
      inuparrow OF hScroll @ call GetCtlValue  -> ctlvalue
      ctlvalue 0= NOT
          IF
         hScroll @ ctlvalue 1- call SetCtlValue
         0 5 ScrollText
          THEN
      ENDOF

      indownarrow OF hScroll @ call GetCtlValue  -> ctlvalue
      ctlvalue maxHs = NOT
          IF
         hScroll @ ctlvalue 1+ call SetCtlValue
         0 -5 ScrollText
          THEN
      ENDOF

      inpageup  OF hScroll @ call GetCtlValue  -> ctlvalue
      ctlvalue 0= NOT
          IF
         hScroll @ ctlvalue 5 - call SetCtlValue
         0 25 ScrollText
          THEN
      ENDOF

      inpagedown OF hScroll @ call GetCtlValue  -> ctlvalue
      ctlvalue maxHs = NOT
           IF
         hScroll @ ctlvalue 5 + call SetCtlValue
         0 -25 ScrollText
           THEN
      ENDOF
   ENDCASE
   hScroll @ call GetCtlValue lastHs !
;

: ControlAction  ( part-code  control-handle -  )
   CASE
       Scroll @ OF DO-Scroll ENDOF
      hScroll @ OF DO-hScroll ENDOF
      swap drop
   ENDCASE
;


\ ***** scrollbar thumb control handler *****

: DO-vThumb ( | ctlV )
   inThumb = IF
   scroll @ call getCtlValue -> ctlV
   lastVs @ ctlV - 5 * 0 scrollText
   ctlV lastVs !
   THEN
;

: DO-hThumb ( | ctlV )
   inThumb = IF
   hscroll @ call getCtlValue -> ctlV
   0 lastHs @ ctlV - 5 * scrollText
   ctlV lastHs !
   THEN
;

: ControlHandler  ( part-code  control-handle -  )
   CASE
       Scroll @ OF DO-vThumb ENDOF
      hScroll @ OF DO-hThumb  ENDOF
      swap drop
   ENDCASE
;


\ ***** initialization *****

: INIT-MBAR
   EditBar ADD
   EditBar APPLEMENU ADD
   APPLEMENU @ "drvr call addresmenu
   EditBar FileMenu  ADD
   EditBar EditMenu  ADD
   EditBar FontMenu  ADD
   Fontmenu @ "font call addresmenu
   Fontmenu @ call countMItems #fonts !
   EditBar SizeMenu  ADD
   EditBar StyleMenu ADD
;

: INIT-TASK
   Editor ADD   ( make the Editor window )
   Editor Scroll ADD   ( add vertical scroll bar )
   Editor hScroll ADD  ( add horizontal scroll bar )
   Editor EditTask BUILD
;

: START-TASK
   ACTIVATE
   ['] ControlAction CAction !
   ['] ControlHandler Control-Vector !
   ['] MBAR-HANDLER MENU-VECTOR !
   BEGIN
       STANDARD-GETFILE
       IF
      Open
      EditFile
      ShutDown
       THEN
       ['] RUN-UPDATE UpdateHook !
       ['] RUN-CONTENT ContentHook !
       ['] RUN-ACTIVATE ActivateHook !
       ['] RUN-CLOSEBOX CloseBoxHook !
       SLEEP STATUS W! ( put Editor task to sleep )
       PAUSE    ( exit this task )
   AGAIN
;

: INIT-EDIT
   INIT-TASK
   INIT-MBAR
   EditBar EditTask MBAR>TASK 4 myStyle w! \ default font,
Monaco
   0 myStyle 2+ w!  \ default face, plain text
   9 myStyle 4 + w! \ default size, 9 point
   0 myStyle 6 + !  \ RGBcolor = ...
   0 myStyle 10 + w! \ ...black
   4   TERect     W!      ( define the text edit rectangle )
   4   TERect 2+  W!
   288 TERect 4 + W!
   503 TERect 6 + W!
   EditTask START-TASK
;


: EDIT    ( wakes up Editor task )
   EDITTASK @
   IF  WAKE STATUS TASK-> EditTask W!
   ELSE   INIT-EDIT
   THEN
;
```

Forth Forum

Graf3D Library Access

Jörg Langowski
MacTutor Editorial Board
MacTutor Vol. 4 No. 6

Graf3D and Mach2 - accessing external libraries

The subject of this month has been initiated by one of our readers, Paul Thomas, who in a desperate mood sent me both a paper letter and GEnie mail, because he needed to know whether it would be possible at all to access the Graf3D routines from Mach2.

A little background for those who weren't with the Mac (or with us, for that matter) from the beginning: Graf3D is a set of routines on top of QuickDraw, which enable to create a 3-d GrafPort with all necessary support to do 3-dimensional perspective drawing. The routines aren't explained in Inside Macintosh, but in several other different documents, one of them being the MPW Pascal manual (Appendix J).

Graf3D was also part of the Lisa Workshop, and one of the first Macintosh demo programs, *Boxes*, used those routines. The Boxes program - many of you might have seen it - draws at random fifteen rectangular boxes on a plane and displays them in a 3-d perspective view. The program is written originally in Lisa Pascal, and also contained on the TML Pascal disk as an example. Listing 4 shows the original program.

Glancing through the code, you might notice that this example certainly wouldn't please the User Interface Thought Police today. Using the whole screen as a GrafPort without regard to anything else on there is a definite no-no in the days of Multifinder, and if you run the example under Multifinder, you won't find your windows back on the screen afterwards because they are all covered by little boxes. Nevertheless, it is a nice example for the use of Graf3D, and we'll show you this month how to translate it into Mach2 (We'll also correct the garbled screen problem under Multifinder).

First and most difficult question: How can we access routines from Forth that are clearly labeled *(Not in ROM)* in the documentation? This remark tells us, of course, that they are part of some object library, no source code available and inaccessible to the Forth user. Just like the printing manager was before it was implemented in the Mach2 and MacForth systems.

There may be other libraries to come, also *not in ROM*, and waiting for updates takes a long time. So I'll give you two general strategies to get access to object library code from Forth, making you independent of those updates.

The first scheme I thought up was simply to write a short Pascal program that calls all the routines at least once - so that they will be loaded -, compile that program, create a linker map if possible, and pass the final application through Nosy. That will get you a more or less annotated assembly listing from which you can extract the source code of the *not in ROM* routines that you are interested in.

Well, that works for you and me on our respective desks, but we certainly cannot publish reverse-engineered Apple source code in MacTutor, at least if we want to stay in business. Also, it is a lot of work (i.e. translating to a different assembler dialect, writing the glue code, etc.), and all in all qualifies as a master example of a dirty hack.

There is an easier way to go, and much more general, too: write some sort of a linker for Pascal library code. We write an assembly main program (Listing 2), which is just a table of JMP instructions to the routines that we want to call, and let the MPW linker do its job (listing 3).

The assembly/link script will create a new resource, **gr3D**, which contains the jump instructions at the beginning, followed by the linked Graf3D code. All we have to do now is to provide a block of memory in our Forth code where this resource can be copied to.

In order to assign Forth words to the JMP instructions, we simply make a table of CREATEs, which will allocate 4 bytes to each name (the length of a JMP d(PC) instruction), and create a dictionary entry so that we know the routine's address in the jump table. After the block of CREATEs, we allocate sufficient memory so that the **gr3D** resource can be loaded (see listing 1).

We write a word, **Init3D**, that gets the **gr3D** resource and copies it into the table, starting with the address of the first Graf3d word, **gInitGrf3D**. Now the Graf3D routines are all set up for use by Mach2. As simple as that.

Well, not quite. We still need to write glue code to move the parameters from the Forth stack (A6) to the Pascal stack (A7). That glue code is rather simple and is just the same as is used for calling toolbox ROM routines. Like the traps, the Graf3D code preserves all registers that are vital to Mach2 (A2-A7, D4-D7). Like in the case of traps, we have to make A7 point to the lowest stack (EXG D4,A7) in order to avoid overwriting of Mach2 space by the intermediate Quickdraw record that is put between the stack and the heap. The glue code for the different Graf3D routines is given at the beginning of listing 1.

The Boxes examples should be self-explanatory when you compare it with the Pascal code; in fact, it might help some of you Pascal programmers and casual Forth readers appreciate that there is really no big difference between Forth and Pascal code...

I should point out some more things. First, the 'access words' for fields in a record, **.x**, **.y**, **.pt1**, and so on, defined at the beginning of the Forth code. Second, the word **restore.screen**, which cleans up the display after the example has run; very important when running under Multifinder. It basically does a PaintBehind(FrontWindow, grayRgn), the same mechanism by which many screen savers restore a blacked-out screen. Third, we

© The Definitive MacTutor, Vol. 4

define a terminal task that runs the Boxes example; that way, we can simply check for **?terminal** after each drawing loop to see whether we're done. Again, we have to make sure (like always in Mach2 multitasking) that the correct GrafPort is set by calling **myPort3D SetPort3D** before each passage through the loop. This is because any **PAUSE**-containing word, such as **?terminal**, may change the active GrafPort.

For using the example, you will first have to create the *Graf3Dglue* resource file with the **gr3D** resource in it, then move that resource into the MACH.RSRC file before starting your Mach2 system. In case you don't have access to MPW, the source code disk contains the *Graf3Dglue* and *MACH.RSRC* files. The complete program, of course, is also on the disk.

That's it for this month; next month I plan to discuss various projects of object-oriented extensions to Mach2 that have recently appeared on the networks. Some of you might also appreciate that NEON hasn't died altogether, but that there is at least one person trying to extend it in a very interesting way.

Listing 1: The Boxes example in Mach2

```
\ Graf3d / Mach2 glue code
\ An example for calling MPW routines from Forth
\ J. Langowski April 1988
\ ---------------------------------------
Only Forth Also Mac Also Assembler

\ some general definitions first
16 CONSTANT portRect

GLOBAL
 CODE SCALE
    MOVE.L   (A6)+,D0
    BMI.S    @1
    MOVE.L   (A6),D1
    ASL.L    D0,D1
    MOVE.L   D1,(A6)
    RTS
@1  MOVE.L   (A6),D1
    NEG.L    D0
    ASR.L    D0,D1
    MOVE.L   D1,(A6)
    RTS
END-CODE

global
CODE white
       MOVE.L (A5),-(A6)
       SUBQ.L #8,(A6)
       RTS
END-CODE MACH

global
CODE black
       MOVE.L (A5),-(A6)
       SUBI.L #16,(A6)
       RTS
END-CODE MACH

global
CODE gray
       MOVE.L (A5),-(A6)
       SUBI.L #24,(A6)
       RTS
END-CODE MACH

: 4ASCII
 0
 4 0 DO
   8 SCALE 0 WORD 1+ C@ +
 LOOP
 ;

4ASCII gr3D CONSTANT 'gr3D \ resource ID

\ Graf3D jump table

CREATE gInitGrf3D
CREATE gOpen3DPort
CREATE gSetPort3D
CREATE gGetPort3D
CREATE gMoveTo2D
CREATE gMoveTo3D
CREATE gLineTo2D
CREATE gLineTo3D
CREATE gMove2D
CREATE gMove3D
CREATE gLine2D
CREATE gLine3D
CREATE gViewPort
CREATE gLookAt
CREATE gViewAngle
CREATE gIdentity
CREATE gScale
CREATE gTranslate
CREATE gPitch
CREATE gYaw
CREATE gRoll
CREATE gSkew
CREATE gTransform
CREATE gClip3D
CREATE gSetPt3D
CREATE gSetPt2D

\ The glue code is 2636 bytes long, so we allocate
\ sufficient additional buffer space to put it into
2600 ALLOT

: Init3D \ gets Graf3D code from gr3D=1 resource
       \ and copies it into buffer
       'gr3D 1 call GetResource
       dup @ swap call SizeRsrc
       ['] gInitGrf3D swap cmove
;

CODE InitGrf3d    ( globalPtr - )
       EXGD4,A7
       MOVE.L (A6)+,-(A7)
       JSRgInitGrf3d
       EXGD4,A7
       RTS
END-CODE

CODE Open3DPort   ( port - )
       EXGD4,A7
       MOVE.L (A6)+,-(A7)
       JSRgOpen3DPort
       EXGD4,A7
       RTS
END-CODE

CODE SetPort3d    ( port - )
       EXGD4,A7
       MOVE.L (A6)+,-(A7)
       JSRgSetPort3d
       EXGD4,A7
       RTS
END-CODE

CODE GetPort3d    ( VAR port - )
       EXGD4,A7
       MOVE.L (A6)+,-(A7)
```


© The Definitive MacTutor, Vol. 4

```
      JSRgGetPort3d
      EXGD4,A7
      RTS
END-CODE

CODE MoveTo2d( x y - )
      EXGD4,A7
      MOVE.L 4(A6),-(A7)
      MOVE.L (A6),-(A7)
      ADDA.W #8,A6
      JSRgMoveTo2d
      EXGD4,A7
      RTS
END-CODE

CODE MoveTo3d( x y z - )
      EXGD4,A7
      MOVE.L 8(A6),-(A7)
      MOVE.L 4(A6),-(A7)
      MOVE.L (A6),-(A7)
      ADDA.W #12,A6
      JSRgMoveTo3d
      EXGD4,A7
      RTS
END-CODE

CODE LineTo2d( x y - )
      EXGD4,A7
      MOVE.L 4(A6),-(A7)
      MOVE.L (A6),-(A7)
      ADDA.W #8,A6
      JSRgLineTo2d
      EXGD4,A7
      RTS
END-CODE

CODE LineTo3d( x y z - )
      EXGD4,A7
      MOVE.L 8(A6),-(A7)
      MOVE.L 4(A6),-(A7)
      MOVE.L (A6),-(A7)
      ADDA.W #12,A6
      JSRgLineTo3d
      EXGD4,A7
      RTS
END-CODE

CODE Move2d  ( dx dy - )
      EXGD4,A7
      MOVE.L 4(A6),-(A7)
      MOVE.L (A6),-(A7)
      ADDA.W #8,A6
      JSRgMove2d
      EXGD4,A7
      RTS
END-CODE

CODE Move3d  ( dx dy dz - )
      EXGD4,A7
      MOVE.L 8(A6),-(A7)
      MOVE.L 4(A6),-(A7)
      MOVE.L (A6),-(A7)
      ADDA.W #12,A6
      JSRgMove3d
      EXGD4,A7
      RTS
END-CODE

CODE Line2d  ( x y - )
      EXGD4,A7
      MOVE.L 4(A6),-(A7)
      MOVE.L (A6),-(A7)
      ADDA.W #8,A6
      JSRgLine2d
      EXGD4,A7
      RTS
END-CODE

CODE Line3d  ( x y z - )
      EXGD4,A7
      MOVE.L 8(A6),-(A7)
      MOVE.L 4(A6),-(A7)
      MOVE.L (A6),-(A7)
      ADDA.W #12,A6
      JSRgLine3d
      EXGD4,A7
      RTS
END-CODE

CODE ViewPort( r - )
      EXGD4,A7
      MOVE.L (A6)+,-(A7)
      JSRgViewPort
      EXGD4,A7
      RTS
END-CODE

CODE LookAt  ( left top right bottom - )
      EXGD4,A7
      MOVE.L 12(A6),-(A7)
      MOVE.L 8(A6),-(A7)
      MOVE.L 4(A6),-(A7)
      MOVE.L (A6),-(A7)
      ADDA.W #16,A6
      JSRgLookAt
      EXGD4,A7
      RTS
END-CODE

CODE ViewAngle   ( angle - )
      EXGD4,A7
      MOVE.L (A6)+,-(A7)
      JSRgViewAngle
      EXGD4,A7
      RTS
END-CODE

CODE Identity
      EXGD4,A7
      JSRgIdentity
      EXGD4,A7
      RTS
END-CODE

CODE Scal ( xfactor yfactor zfactor - )
      EXGD4,A7
      MOVE.L 8(A6),-(A7)
      MOVE.L 4(A6),-(A7)
      MOVE.L (A6),-(A7)
      ADDA.W #12,A6
      JSRgScale
      EXGD4,A7
      RTS
END-CODE

CODE Translate   ( dx dy dz - )
      EXGD4,A7
      MOVE.L 8(A6),-(A7)
      MOVE.L 4(A6),-(A7)
      MOVE.L (A6),-(A7)
      ADDA.W #12,A6
      JSRgTranslate
      EXGD4,A7
      RTS
END-CODE

CODE Pitch   ( xangle - )
      EXGD4,A7
      MOVE.L (A6)+,-(A7)
      JSRgPitch
      EXGD4,A7
      RTS
END-CODE

CODE Yaw  ( yangle - )
      EXGD4,A7
      MOVE.L (A6)+,-(A7)
      JSRgYaw
      EXGD4,A7
      RTS
END-CODE

CODE Rol  ( zangle - )
      EXGD4,A7
      MOVE.L (A6)+,-(A7)
      JSRgRoll
      EXGD4,A7
      RTS
END-CODE

CODE Skew ( zangle - )
      EXGD4,A7
      MOVE.L (A6)+,-(A7)
      JSRgSkew
      EXGD4,A7
      RTS
END-CODE

CODE Transform   ( src dst - )
      EXGD4,A7
      MOVE.L 4(A6),-(A7)
      MOVE.L (A6),-(A7)
      ADDA.W #8,A6
      JSRgTransform
      EXGD4,A7
      RTS
END-CODE

CODE Clip3D  ( src1 src2 dst1 dst2 - flag
)
      EXGD4,A7
      CLR.W  -(A7)
      MOVE.L 12(A6),-(A7)
      MOVE.L 8(A6),-(A7)
      MOVE.L 4(A6),-(A7)
      MOVE.L (A6),-(A7)
      ADDA.W #16,A6
      JSRgClip3D
      MOVE.W (A7)+,D0
      EXT.L  D0
      MOVE.L D0,-(A6)
      EXGD4,A7
      RTS
END-CODE

CODE SetPt3d ( pt3D x y z - )
      EXGD4,A7
      MOVE.L 12(A6),-(A7)
      MOVE.L 8(A6),-(A7)
      MOVE.L 4(A6),-(A7)
      MOVE.L (A6),-(A7)
      ADDA.W #16,A6
      JSRgSetPt3D
      EXGD4,A7
      RTS
END-CODE

CODE SetPt2d ( pt2D x y - )
      EXGD4,A7
      MOVE.L 8(A6),-(A7)
      MOVE.L 4(A6),-(A7)
      MOVE.L (A6),-(A7)
```

```
      ADDA.W #12,A6
      JSR gSetPt2D
      EXG D4,A7
      RTS
END-CODE

\ Translation of Boxes.pas example into Forth follows
\ ---------------------------------------------

: .x ; mach
: .y 4 + ; mach
: .z 8 + ; mach

: .pt1 ; mach
: .pt2 12 + ; mach

15 CONSTANT BoxCount
Variable MyPort       104 vallot
            \ grafPort is 108 bytes long
Variable MyPort3D 150 vallot
            \ graf3DPort is 154 bytes long
Variable boxArray   24 BoxCount * vallot
            \ BoxCount * 2* point3D @ 3 long words
Variable nboxes               \ # of boxes made
Variable MyBox    20 vallot \ 24 bytes for 2* point3d
Variable p1         8 vallot \ point3d
Variable p2         8 vallot \ point3d
Variable myRect     4 vallot \ Rect
Variable testRect   4 vallot \ Rect

: DrawBrick ( pt1 pt2 | tempRgn - )
      call NewRgn -> tempRgn

      call OpenRgn
      pt1 .x @ pt1 .y @ pt1 .z @ MoveTo3D
      pt1 .x @ pt1 .y @ pt2 .z @ LineTo3D
      pt2 .x @ pt1 .y @ pt2 .z @ LineTo3D
      pt2 .x @ pt1 .y @ pt1 .z @ LineTo3D
      pt1 .x @ pt1 .y @ pt1 .z @ LineTo3D
      tempRgn call CloseRgn
      tempRgn white call FillRgn

      call OpenRgn
      pt1 .x @ pt1 .y @ pt2 .z @ MoveTo3D
      pt1 .x @ pt2 .y @ pt2 .z @ LineTo3D
      pt2 .x @ pt2 .y @ pt2 .z @ LineTo3D
      pt2 .x @ pt1 .y @ pt2 .z @ LineTo3D
      pt1 .x @ pt1 .y @ pt2 .z @ LineTo3D
      tempRgn call CloseRgn
      tempRgn gray call FillRgn

      call OpenRgn
      pt2 .x @ pt1 .y @ pt1 .z @ MoveTo3D
      pt2 .x @ pt1 .y @ pt2 .z @ LineTo3D
      pt2 .x @ pt2 .y @ pt2 .z @ LineTo3D
      pt2 .x @ pt2 .y @ pt1 .z @ LineTo3D
      pt2 .x @ pt1 .y @ pt1 .z @ LineTo3D
      tempRgn call CloseRgn
      tempRgn black call FillRgn

      white call penpat
      pt2 .x @ pt2 .y @ pt2 .z @ MoveTo3D
      pt2 .x @ pt2 .y @ pt1 .z @ LineTo3D
      pt2 .x @ pt1 .y @ pt1 .z @ LineTo3D
      call pennormal

      tempRgn call DisposRgn
;

: hi -16 scale ;

: chkBox ( | box - )
      1 ( flag )
      nBoxes @ 0 DO
         boxArray i 24 * + -> box
         testRect
         box .pt1 .x @ hi
         box .pt1 .y @ hi
         box .pt2 .x @ hi
         box .pt2 .y @ hi  call SetRect
         testRect -1 -1 call InSetRect
         myRect testRect testRect call SectRect
         IF drop 0 leave THEN
      LOOP
;

: MakeBox ( | p1x p1y p1z p2x p2y p2z box ii - )
      call random 70 mod 15 - 1 call FixRatio -> p1x
      call random 70 mod 10 - 1 call FixRatio -> p1y
      0 -> p1z

      call random 30 mod abs 10 + 1 call FixRatio p1x +
         -> p2x
      call random 45 mod abs 10 + 1 call FixRatio p1y +
         -> p2y
      call random 30 mod abs 10 + 1 call FixRatio p1z +
         -> p2z

      myRect p1x hi p1y hi p2x hi p2y hi call SetRect

      chkBox IF
         p1x myBox .pt1 .x !
         p1y myBox .pt1 .y !
         p1z myBox .pt1 .z !
         p2x myBox .pt2 .x !
         p2y myBox .pt2 .y !
         p2z myBox .pt2 .z !

         0 -> ii
         myBox boxArray nBoxes @ 24 * + 24 cmove

         BEGIN
         myBox .pt1 .y @  boxArray ii + .pt2 .y @ >
         myBox .pt2 .y @  boxArray ii + .pt1 .y @ > and
         myBox .pt1 .x @  boxArray ii + .pt2 .x @ <
         myBox .pt2 .x @  boxArray ii + .pt1 .x @ < and or
         WHILE
            24 +> ii
         REPEAT
         ii 24 / -> ii

         ii 1+ nBoxes @ DO
            boxArray i 1- 24 * +
            boxArray i 24 * +
            24 cmove
         -1 +loop
         myBox boxArray ii 24 * + 24 cmove

         1 nBoxes +!
      THEN
;

: drawGrid
      11 -10 DO
         i 10 * 1 call FixRatio -100 1 call FixRatio 0
         MoveTo3D
         i 10 * 1 call FixRatio  100 1 call FixRatio 0
         LineTo3D
      LOOP

      11 -10 DO
         -100 1 call FixRatio i 10 * 1 call FixRatio 0
         MoveTo3D
          100 1 call FixRatio i 10 * 1 call FixRatio 0
         LineTo3D
      LOOP
;
```


© The Definitive MacTutor, Vol. 4

```
: restore.screen
      call drawmenubar
      call frontwindow
      grayrgn @ call paintbehind
      call showcursor
;

: main
      init3d
      call hidecursor
      myPort call OpenPort
      myPort3D Open3DPort
      myPort portRect + ViewPort
      -100 1 call FixRatio  75 1 call FixRatio
       100 1 call FixRatio -75 1 call FixRatio
         LookAt
      30 1 call FixRatio ViewAngle
      Identity
      20 1 call FixRatio Rol
      70 1 call FixRatio Pitch

      BEGIN
         myPort3D SetPort3D
         0 nBoxes !
         BEGIN makeBox nBoxes @ boxCount = UNTIL

         white call penPat
         black call backPat
         myPort portRect + call EraseRect

         drawGrid

         0 nBoxes @ 1- DO
            boxArray i 24 * +
                  dup .pt1 swap .pt2 DrawBrick
         -1 +LOOP

      ?terminal UNTIL
      restore.screen
      bye
;

NEW.WINDOW Boxes
" Boxes" Boxes TITLE
0 0 20 20 Boxes BOUNDS
Plain Visible NoCloseBox NoGrowBox Boxes ITEMS

600 5000 terminal Box

: go.box activate main ;

: start
      Boxes add
      Boxes Box build
      Box go.Box
;

.( To create a turkey application ) cr
.( type TURNKEY START BOXES )
```

Listing 2: MPW Assembly glue code for accessing the Graf3D Pascal library

```
                TITLE    'Graf3D glue code for Mach2'
                BLANKS       OFF
                PRINT            OFF
                INCLUDE      'Traps.a'
                INCLUDE      'ToolEqu.a'
                INCLUDE      'QuickEqu.a'
                INCLUDE      'SysEqu.a'
                INCLUDE      'Graf3DEqu.a'
                PRINT            ON

GrafGlue        MAIN
      JMP   InitGrf3D
      JMP   Open3DPort
      JMP   SetPort3D
      JMP   GetPort3D
      JMP   MoveTo2D
      JMP   MoveTo3D
      JMP   LineTo2D
      JMP   LineTo3D
      JMP   Move2D
      JMP   Move3D
      JMP   Line2D
      JMP   Line3D
      JMP   ViewPort
      JMP   LookAt
      JMP   ViewAngle
      JMP   Identity
      JMP   Scale
      JMP   Translate
      JMP   Pitch
      JMP   Yaw
      JMP   Roll
      JMP   Skew
      JMP   Transform
      JMP   Clip3D
      JMP   SetPt3D
      JMP   SetPt2D
      END
```

Listing 3: MPW assembly / link script for generating the gr3D resource (to be put into the MACH.RSRC file)

```
      Asm Graf3dglue.a -l
      Link Graf3dglue.a.o ∂
      {Libraries}Interface.o ∂
      -rt gr3D=1 -o Graf3dglue -l > Graf3dglue.map
```

Listing 4: The original Boxes example (from TML Pascal)

```
      PROGRAM Boxes;

{ Boxes demonstrates the use of the Three Dimensional
Quickdraw package Graf3D.

  Pascal source:  Boxes.Pas
  Resources:      *none*

  This program is from Apple's Lisa Software Supplement,
modified for TML Systems Pascal compiler.
}

USES MacIntf, FixMath, Graf3D;

CONST  boxCount = 15;
       keyOrMouse = 10;

TYPE   Box3D =
           RECORD
          pt1: Point3D;
          pt2: Point3D;
           END;

VAR    myPort:   GrafPort;
       myPort3D: Port3D;
       boxArray: ARRAY [0..boxCount] OF Box3D;
       nBoxes:   INTEGER;
       i:        INTEGER;
       dummy:    EventRecord;

PROCEDURE DrawBrick(pt1, pt2: Point3D);
{ draws a 3D brick with shaded faces.
```

```
  only shades correctly in one direction.
}
VAR    tempRgn: RgnHandle;
BEGIN
   tempRgn := NewRgn;
   OpenRgn;
   MoveTo3D(pt1.X, pt1.Y, pt1.Z); { front face, y=y1 }
   LineTo3D(pt1.X, pt1.Y, pt2.Z);
   LineTo3D(pt2.X, pt1.Y, pt2.Z);
   LineTo3D(pt2.X, pt1.Y, pt1.Z);
   LineTo3D(pt1.X, pt1.Y, pt1.Z);
   CloseRgn(tempRgn);
   FillRgn(tempRgn, white);

   OpenRgn;
   MoveTo3D(pt1.X, pt1.Y, pt2.Z); { top face, z=z2 }
   LineTo3D(pt1.X, pt2.Y, pt2.Z);
   LineTo3D(pt2.X, pt2.Y, pt2.Z);
   LineTo3D(pt2.X, pt1.Y, pt2.Z);
   LineTo3D(pt1.X, pt1.Y, pt2.Z);
   CloseRgn(tempRgn);
   FillRgn(tempRgn, gray);

   OpenRgn;
   MoveTo3D(pt2.X, pt1.Y, pt1.Z); { right face, x=x2 }
   LineTo3D(pt2.X, pt1.Y, pt2.Z);
   LineTo3D(pt2.X, pt2.Y, pt2.Z);
   LineTo3D(pt2.X, pt2.Y, pt1.Z);
   LineTo3D(pt2.X, pt1.Y, pt1.Z);
   CloseRgn(tempRgn);
   FillRgn(tempRgn, black);

   PenPat(white);
   MoveTo3D(pt2.X, pt2.Y, pt2.Z); { outline right }
   LineTo3D(pt2.X, pt2.Y, pt1.Z);
   LineTo3D(pt2.X, pt1.Y, pt1.Z);
   PenNormal;

   DisposeRgn(tempRgn);
END;

PROCEDURE MakeBox;
LABEL  1;
VAR    myBox: Box3D;
       i, j, h, v: INTEGER;
       p1, p2: Point3D;
       myRect: Rect;
       testRect: Rect;
BEGIN
   p1.x := FixRatio((Random mod 70 - 15), 1);
   p1.y := FixRatio((Random mod 70 - 10), 1);
   p1.z := 0;

   p2.x := p1.x + FixRatio((10 + ABS(Random) MOD 30), 1);
   p2.y := p1.y + FixRatio((10 + ABS(Random) MOD 45), 1);

   p2.z := p1.z + FixRatio((10 + ABS(Random) MOD 35), 1);

   { reject box if it intersects one already in list }
    SetRect(myRect, HiWord(p1.x), HiWord(p1.y), HiWord(p2.x),
HiWord(p2.y));

   FOR i := 0 TO nBoxes - 1 DO BEGIN
      WITH boxArray[i] DO
       SetRect(testRect,
              HiWord(pt1.x),
          HiWord(pt1.y),
          HiWord(pt2.x),
              HiWord(pt2.y));
      InSetRect(testRect, -1, -1);
      IF SectRect(myRect, testRect, testRect) THEN goto 1
      END;

   myBox.pt1 := p1;
   myBox.pt2 := p2;

   { calc midpoint of box and its distance from the eye }
   i := 0;
   boxArray[nBoxes].pt1 := myBox.pt1; { sentinel }
   boxArray[nBoxes].pt2 := myBox.pt2;

   WHILE
     ((myBox.pt1.y > boxArray[i].pt2.y) AND (myBox.pt2.y >
boxArray[i].pt1.y)) OR
     ((myBox.pt1.x < boxArray[i].pt2.x) AND (myBox.pt2.x <
boxArray[i].pt1.x)) DO
      inc(i); { insert in order of dist }
   FOR j := nBoxes DOWNTO i + 1 DO boxArray[j] := boxArray[j - 1];
   boxArray[i] := myBox;
   inc(nBoxes);
1:
END;

BEGIN { main program }
   FlushEvents(everyEvent, 0);
   InitGraf(@thePort);
   InitCursor;
   HideCursor;
   OpenPort(@myPort);
   Open3DPort(@myPort3D);

   { put the image in this rect }
   ViewPort(myPort.portRect);

   { aim the camera into 3D space }
   LookAt(FixRatio(-100, 1),
          FixRatio(75, 1),
         FixRatio(100, 1),
          FixRatio(-75, 1));

   { choose lens focal length }
   ViewAngle(FixRatio(30, 1));
   Identity;
   Roll(FixRatio(20, 1));
   Pitch(FixRatio(70, 1)); { roll and pitch the plane }

   REPEAT
      nBoxes := 0;
      REPEAT
         MakeBox
      UNTIL nBoxes=boxCount;

      PenPat(white);
      BackPat(black);
      EraseRect(myPort.portRect);

      FOR i := -10 TO 10 DO BEGIN
         MoveTo3D(FixRatio(i*10, 1), FixRatio(-100, 1), 0);
         LineTo3D(FixRatio(i*10, 1), FixRatio(100, 1), 0);
         END;

      FOR i := -10 TO 10 DO BEGIN
         MoveTo3D(FixRatio(-100, 1), FixRatio(i*10, 1), 0);
         LineTo3D(FixRatio(100, 1), FixRatio(i*10, 1), 0);
         END;

      FOR i := nBoxes-1 DOWNTO 0 DO
         DrawBrick(boxArray[i].pt1,boxArray[i].pt2);

   UNTIL OSEventAvail(keyOrMouse, dummy);
END.
```

Forth Forum

Objective Activities in Forth

Jörg Langowski
MacTutor Editorial Board
MacTutor Vol. 4 No. 7

Objects in Forth, International Resources & A Priority TaskScheduler

One of Forth's great advantages is its extensibility. Since the Forth compiler is just made up from other Forth words, and the data structures into which each new word is compiled are very well documented and accessible to the user, you can very easily rewrite the compiler and thereby define your own language. (These comments are obvious to the seasoned Forth user; I've just repeated them to emphasize the difference to the more 'canonical' languages like Pascal or C).

With a language as flexible as Forth (very much like LISP in that aspect), it is no surprise that many have set out to create their own language on top of a Forth kernel. Famous examples are the object-oriented extensions ForthTalk and, of course, NEON. While ForthTalk (which has been covered by Paul Snively in MacTutor V3#2) seems to be alive and well, and still offered by Creative Solutions in their newsletter, NEON seems to have been all but abandoned by their creator, Kriya Systems. After the 2.0 update, more than a year ago, no more NEON news. And it was such an elegant implementation of object-oriented programming! Judged from bulletin board posts and letters I receive every now and then, many of you must be more than disappointed at the slow disappearance of NEON from the scene of Macintosh languages.

It is about a recent attempt to extend and -maybe- resurrect NEON, and about several other projects of object-oriented extensions to Mach2 Forth, that I wish to report this month. A lot of the information I am presenting comes from discussions and postings on the Mach2 roundtable on GEnie; therefore this column is, again, somewhat biased towards Mach2. MacForth users, please, read on and comment; a standard for an object oriented Forth would be appreciated by most of us.

Object Orientation in Forth

Why would one choose Forth to implement an object-oriented language? Besides the built-in extensibility of the Forth compiler, some aspects of object orientation are already contained in Forth. As a reminder, an object is a structure that contains not only data, but also procedures: *methods*, which can operate on the contents of the object or on other data. Example: an array of integer numbers which, given an index n, automatically returns the address of the nth element. The classical Forth definition of a 'defining word' for arrays with 32 bit-size elements is:

```
: ARRAY  CREATE ( # - ) 4 * ALLOT
( reserves # cells in the dictionary )
DOES>  ( n <adr> - adr ) SWAP 4 * + ;
```

The definition of an array of 100 long integer numbers would then be `100 ARRAY myArray`, while `47 myArray` puts the address of the 47th element of myArray on the stack. When the array is first defined, the **CREATE** part of the definition is executed, and space is reserved in the dictionary for the array's contents. In object-oriented lingo, this would be called creation of an 'instance' `myArray` of the 'class' `ARRAY`. `myArray`'s 'instance variables' would correspond to the space reserved in the dictionary for the 100 32-bit numbers, and its one and only 'method' would be the **DOES>** part of the definition, which is invoked when the word `myArray` is executed.

If you now extend the definition `ARRAY` like this:

```
: ARRAY  CREATE 4 * ALLOT
    DOES>  swap  ( n <adr> msg - adr )
        CASE
            1 OF ( addr: ) SWAP 4 * + ENDOF
            2 OF ( at: ) SWAP 4 * + @ ENDOF
            3 OF ( to: ) SWAP 4 * + ! ENDOF
;
```

and define

```
: addr: 1 ;
: at: 2 ;
: to: 3 ;
```

`47 addr: myArray` would return the address of the 47th element, `47 at: myArray` would return its value, and `3 47 to: myArray` would store 3 in that cell.

`myArray` can therefore be viewed as an 'object' to which you can send 'messages' that determine the method used on its contents. `myArray` has three methods defined, `addr:`, `at:`, and `to:`, which have the method selectors 1, 2 and 3.

This example showed you one aspect of object-orientation, *encapsulation* of the data structures: you access the object's data by an interface given by the messages that are sent to it. No other access would be possible (although in this case it is easy to circumvent that restriction, if one does it deliberately). The other aspect - not covered by the example - is *inheritance* of methods and data structures: we also need super- and subclasses, automatic message passing to classes further up in the hierarchy, method override, and so on.

Also, the example I gave resolves method references at run time (*'late binding'*). This takes time, and if message and object are already defined at compile time, the compiler should resolve the reference so that the code generated for `i at: myArray` does

not take any longer than a simple indexed fetch (like `myArray i 4 * + @`). This case is called *'early binding'*.

Let me now describe the three projects of object-oriented extensions for Mach2 that I'm aware of and their basic differences.

Object Oriented Extensions for Mach2

One attempt to create a NEON-like environment to Mach2 came from Aleksey Novicov, of Palo Alto Shipping, who developed a preliminary version of his object oriented system, called OPEX, whose syntax is very NEON-like. Following parts of Aleksey's description of OPEX:

Brief Introduction to OPEX (v0.50): An Object-oriented Programming Extension for MACH 2 Aleksey Novicov

...OPEX contains all of the elements of an object-oriented programming environment. This includes the ability to define CLASSES, public OBJECTS of various classes, INSTANCE VARIABLES, and METHODS that are called by SELECTORS. Also, a class can be defined as having a SUPER CLASS,and thus inherit all of the data structures and methods of that super class. *All of the method/object binding is currently static only (i.e., only compile-time binding, no run-time binding yet).*

...Two new vocabularies are present called OBJECTS and IVARS. In normal operation no special consideration need be given to these vocabularies. However, if any public objects are to accessed, then the OBJECTS vocabulary must always be present in the search order.

...This a list of words that currently are available in OPEX...:

```
:CLASS      ;CLASS          ;M              :M
BYTES       <SUPER          OBJECT          MW!
MW@         SEGMENT     PERMANENT       INITALL
(abs)
```

Also, the following MACH 2 words have been enhanced to work with OPEX:

```
WORKSPACE       NEW-SEGMENT         TURNKEY
```

There are many things that still need to be implemented. Some of the obvious include dynamic binding, support of variable length ARRAYs, and dynamic object allocation on the heap. Also, the special objects, SELF and SUPER, still need to be implemented.

More extensive error handling will be implemented to aid the programmer. Hopefully, OPEX will become more robust as time goes on without sacrificing speed. Whatever the case however, it will still run quite fast by virtue of the subroutine-threaded FORTH that it was built upon.

The current version does not yet support multi-tasking. However, any programs written with OPEX that use a single task will work fine." —Aleksey Novicov

In the following, some examples of actual OPEX code which shows the NEON-like syntax:

```
:CLASS INT <SUPER Object
    2 BYTES

    :M PUT:  MW!  ;M
    :M GET:  MW@  ;M
;CLASS

\ Class definitions must start with :CLASS, end with
\ ;CLASS, and have a SUPER class.  Methods definitions
\ must begin with :M and end with ;M.  Methods names must
\ always end with a colon. MW! and MW@ respectively
\ store and fetch a 16-bit integer to and from the object and
\ data area that is pointed to by the topmost value on the
\ object stack.

:CLASS Point <SUPER Object
    INT x
    INT y

    :M PUT:  PUT: y PUT: x  ;M
    :M GET:  GET: x GET: y  ;M
;CLASS


\ Selector or methods names can be reused without worry.
\ The method of the class of a particular object will always
\ be correctly called.


:CLASS REGION <SUPER Object
    ( for use with QuickDraw regions )
    LongINT RgnHandle

    :M OPEN:   CALL NewRgn  RgnHandle !
                            CALL OpenRgn  ;M
    :M CLOSE:   RgnHandle @  CALL CloseRgn  ;M
    :M DRAW:   RgnHandle @  CALL FrameRgn  ;M
    :M PAINT:   RgnHandle @  CALL PaintRgn  ;M
    :M ERASE:   RgnHandle @  CALL EraseRgn  ;M
    :M INVERT:   RgnHandle @  CALL InvertRgn  ;M

    :M WITHIN:    ( `location - flag )
        @  RgnHandle @  CALL PtInRgn  ;M

    :M MOVE: ( dh dv - )
        RgnHandle @  dh dv CALL OfSetRgn  ;M
;CLASS

\ In all of the above methods "GET: RgnHandle" could've
\ been used instead of "RgnHandle @" and
\ "PUT: RgnHandle" could've been used instead of
\ "RgnHandle !".  The advantage to accessing instance
\ variables in this manner is speed.  Also note that any
\ method can have its own set of local variables (see
\ MOVE: method).

POINT thisPt
POINT thatPt
INT x INT y

3 put:  x   4 put: y
get: x  get: y  put: thisPt ( set thisPt to some value )
get: thisPt swap put: thatPt  ( swap x and y for thatPt )
```

OPEX seems like one good step forward on the way to a

NEON-like environment in Mach2, even though some things still need to be implemented. The discussion on OPEX has been pretty inactive for the last few months; I encourage all of you to contact the author or Palo Alto Shipping on the GEnie RoundTable to make this project advance.

The second OOPS-Forth that can be found on GEnie has been written by Jim Straus. His package is much less NEON-like. In his own words:

OOPS-Forth
Jim Straus

"It is based on an article by Dick Pountain (of Byte fame) that appeared in the Journal of Forth Applications and Research (volume 3, number 3). The source code given in the article was modified to work with Mach2. This project was started because I've had a long standing interest in object oriented programming and I wanted an object oriented language where I could see what was going on inside. I started from the article by Dick Pountain so that I could see how someone else had done it. I figured that it would be easier to start with a basic outline for the extensions. While implementing this package, I have come across many things that I would do differently. Some of these include removing distinction between classes and instances, allow for "early binding", and making the code for looking up method selectors much more efficient. I hope to get around to doing a second version, but I thought that people might want to try out an object oriented language in the mean time.

Included in this package are the files: 'Object Oriented Forth-Mach', which contains the defining words to create new classes and the definition of the root class %OBJECT. 'Graphic Objects' contains some demonstration classes and instances of those classes. The graphical objects are points, pens (which understand some Quick Draw messages, such as PenPat and many Logo turtle-like messages, such as right: and go:).

...

In an object oriented language you send messages to objects. The general syntax in this implementation is: <arguments> object \ methodselector

[different from the NEON syntax, which would be <arguments> selector: object—JL]

where the <arguments> depend on the method selected and "\" is the word that sends the message to "object" telling it to look for the method associated with "methodselector" and to execute the code for that method. In this implementation both classes and instances of classes are objects and may be sent messages. Classes define both the methods to which the class will responds and also the methods to which instances of that class will respond. *[This is also different from NEON, where classes aren't objects.—JL]*

New classes inherit methods from their superclass. For example, the class %polygon defines its own methods, but polygons also recognize the methods of their superclass %pen. Also note that the class %polygon can override the methods in %pen to either replace or enhance them. The top of this hierarchy of classes is always the class %OBJECT. %OBJECT is special in that it handles creating new instances of a class, handles unrecognized method selectors and provides a general framework for what methods every object understands.

An instance of a class contains a pointer to its class and contains the actual storage needed by the instance. This allows each instance to have different values in its variables. For example, the two polygons "fred" and "sam" each understand all of the same methods, but when each is asked to "\ draw" itself, they display as different shapes and at different locations. This is because their values are different.

...

An object leaves its address on the stack. The word "\" (redefined from the Mach2 line commentor) calculates a hash value for the word following it (the method selector) and sends that value to the object. The object looks up the method selector with a case statement and executes its code. Any values needed by the method should be left on the stack beforehand. For example, "fred\draw" would cause fred to draw itself and "10 bic \go:" would cause bic to move 10 steps in whatever direction it was headed." —Jim Strauss

Here is an empty class definition:

```
CLASS:-  classname
SUPERCLASS:- %OBJECT

    ENDCLASSVARS;

    CLASS METHODS:-
    ENDMETHODS;

    ENDINSTVARS;

    INSTANCE METHODS:-
    ENDMETHODS;
ENDCLASS;
```

A code fragment written in this dialect would look like:

```
CLASS:- %polygon
SUPERCLASS:- %pen

    ENDCLASSVARS;

    CLASS METHODS:-
    ENDMETHODS;

    INT INSTVAR SIDES ( number of sides in polygon )
    INT INSTVAR LENGTH
                ( the length of the sides of the polygon )
    ENDINSTVARS;

    INSTANCE METHODS:-
        ~ init   :: super\ init 20 LENGTH ! 4 SIDES ! ;;
        ~ print  :: SIDES @ . LENGTH @ .
                    super\ print
                    ." \ length: \ sides: " ;;
        ~ =      :: self \ class OVER \ related?
                    IF DUP super\ =
                      OVER \ sides SIDES @ = AND
                      SWAP \ length LENGTH @ = AND
                    ELSE DROP 0 THEN ;;
        ~ draw   :: 168 ( 360d) SIDES @ / ( find angle )
                    SIDES @ 0 DO
                      LENGTH @ super\ go: DROP
```

```
                    DUP self \ right: DROP
                   LOOP DROP self ;;
        ~ go:       :: self \ length: \ draw ;;
        ~ sides:    :: SIDES ! self ;;
        ~ sides     :: SIDES @ ;;
        ~ length:       :: LENGTH ! self ;;
        ~ length    :: LENGTH @ ;;
        ~ times:    :: 168 ( 360d) OVER /
                       SWAP 0 DO
                           self \ draw DROP
                           DUP self \ right: DROP
                           LOOP DROP self ;;
    ENDMETHODS;

ENDCLASS;
DECIMAL

call frontwindow call setport
100 100 %pen \ setcenter DROP
                ( initialize the class %pen )
0 integer bic     ( create a variable for a new object )
%pen \ new bic to ( an instance of class %pen )
0 integer fred        ( a variable for a new box )
%polygon \ new fred to
                ( an instance of class %polygon )
0 integer sam
%polygon \ new sam to ( create another polygon )
0 integer poly
%polygon \ new poly to( create another polygon )
```

" "

This example - although incomplete - may give you an impression how OOPS Forth code looks like in Jim Straus' system. I'll leave it with that; for a complete description and the source code you should check on the Mach2 roundtable on GEnie.

The third Mach2-OOPS, written by Wayne Joerding from Pullman, WA, is similar to the one just described in that it uses class defining objects (CDOs) to create instances or subclasses of a class. Wayne has prepared, and almost finished, a big article on his implementation which we'll hopefully print in one of the next issues. The implementation allows early and late binding. Again, a code fragment probably serves best to explain the syntax:

```
\ ====== Variable size Array Class =========
Integer Define.Child.Class
    :Instance
        I.Var    Max.Index
            \ max size of array, for error checking
        I.Var    Length
            \ number of current elements in array,
            \ measured by cells
        I.Pntr   Start
            \ points to the start of array memory
    Hide
        :M Describe
            cr
             ." — Instance Information ———"
            Name
            cr ." Max Length in cells  = "
                Max.Index @ 1+ .
            cr ." Cell size in bytes   = " Int @ .
            Pr.Imeths
            ;
        :M Store ( x i - )
        \ Store value x in array for index = i,
        \  first cell has index of zero
            Max.Index @ over < over 0 < or IF
                ." index out of bounds" abort THEN
                \ <- error checking
            Int @ * Start + ! ;
        :M Retrieve ( i - )
                \ Retrieve value of array for index = i.
            Max.Index @ over < over 0 < or IF
                ." index out of bounds" abort THEN
                \ <- error checking
            Int @  * Start + @    ;
    ;Instance

    :Class
        :M Make.Instance ( n c - ) ( <name> -IN- )
        \ Array instance of size n cells, cell size of c
            CREATE immediate
            ins.Key @
            ,  \ store key to methods of parent class
            dup ,        \ save cell size in 'Int' variable
            over 1-  ,   \ make and save Max.Index
                         \ of array for error checking
            0 ,          \ init current Length to zero
            * ins.Size @ + allot
            DOES> dup @ Selector  ( ins.PFA - ins.PFA key )

            ;
    ;Class
Integer  Name.Child.Class      Array    ( n c - )
```

You notice the strategy: The class INTEGER is just another object to which we can send the message Define.Child.Class, followed by a subclass definition; right after that definition, we send the message Name.Child.Class, followed by the subclass name. A bit unusual for those used to the NEON syntax, but it works very well. Typical code would look like:

```
\ - Array test —
100 Array Make.Instance myArray
3 47 myArray Store
47 myArray  Retrieve . cr

\ - test of String —
20 String Make.Instance StrHello
: hel ." This is StrHello" ;
' hel 4 + StrHello Store
: chk.Str0b StrHello Print ;
```

I hope you are anxious to see the actual article, which should follow soon.

NEON - still breathing (?!)

NEON's problem has been - I think - to have been on the market too early; there were just too may bugs left in the first releases. NEON 2.0 always behaved quite nicely in my hands; others said it still crashed very often. Anyway, looking back one could perhaps consider NEON 2.0 the first mature release, and it is a shame that there has been only silence from Kriya Systems for so long.

Extending Neon
Jim Savidge

Recently, an avid NEON user, Jim Savidge, has taken the NEON kernel and built its own extensions on top of it, making it into an even more powerful system than NEON itself. I will give you a quick description of his enhancements.

(Excerpt from GEnie)
Category 2, Topic 14
Message 47 Sat Jan 02, 1988
J.T.SAVIDGE at 23:41 CST

For those who are interested I have made some very interesting and powerful extensions to Neon that might have some use in a Mach 2 implementation. I am at the moment trying to patch Neon to work on a Mac II. Now that Neon is no longer on the public market and that it uses self-modifying code etc.., wish me luck. I rather have my extensions in use in an expanding market. (hint hint Mach users) Here is a simple list of my extensions:

1) Multiple-inheritance w/ no limit to number of super classes, User can pick which class a message will be sent to, and which particular class variable it is sent to, (of the classes many superclasses.)

2) Private methods. The message name will no longer be recognized after the class definition is close off.

3) Local Procedures. At the Forth level. This allows a local Forth word, (a Forth definition inside a Forth definition) to access the local variables.

4) A multi-window, single-stepping debugger, with multiple levels of break-pointing. (i.e. breakpoint to test a word, pop into debugger while testing, breakpoint test, continue last break, then even go back to where you were on the first breakpoint, etc..) Windows include: trace window, Tib window with In indicator, Stack window that shows decompiled meanings of Parameter, Return, and Methods stacks.

5) Unique-methods. Creating a single object with methods of its own.

6) Clone-Object and Copy-Object words.

7) Echo-to-disk

8) Saving off of Method and Ivar names for later Decompiling/ Debugging.

9) Other "minor improvements".

If anybody would like me to contribute to an object-extension to Mach 2 or in creating any new Forth/Object language *PLEASE* contact me.

James T. Savidge, Saturday, January 2, 1988.

Even for those not directly interesting in object-oriented programming, the definitions for local procedures and the multi-window debugger should be very worthwhile enhancements. For the object-oriented part, the multiple inheritance - allowing a new subclass to have several superclasses - is, of course, especially interesting.

A message to an object that has several superclasses, and which cannot be handles by the object itself, is first passed to the superclass which is defined first, if the superclass cannot handle it, it is passed further upward in the hierarchy, until the class OBJECT. If the method still can't be found, the message is passed to the superclass which was defined second, and so on. An example of a class definition in Jim Savidge's enhanced NEON would look like:

```
:Class Qqq <Super Rrr
           <Super Sss
     Var Qqqvar

    :M Aaa: ." Qqq Message " ;M

;Class

Qqq myObj
```

The message **Aaa:** can be handled directly by objects of class **Qqq**, while other messages are first passed to class **Rrr**, then to Sss. One more specialty is *class picking*; if you send **Sss.Xxx: myObj**, the message **Xxx:** will be passed directly to the superclass Sss.

Private methods which are only accessible from within the class definition can be defined through the following construct:

```
:CLASS FIRST <SUPER OBJECT

    :PRIVATE
        :M AAA: ." AAA " ;M
        :M BBB: ." BBB " ;M
    ;PRIVATE

    :M CCC: ." CCC " aaa: self ;M
    :M DDD: ." DDD " bbb: self ;M

;CLASS

FIRST  ONE
```

In this case, sending **ccc: one** would print 'CCC AAA', while **aaa: one** would result in an error message. A new level of information hiding which can be very useful.

I have tested all of the above, and more, on my MacII, and to my great surprise (I hadn't tested NEON on the MacII yet) discovered that NEON 2.0 still runs on a MacII, even under Multifinder! It shouldn't run, though, since my cache was turned on, and trap calls in NEON use self-modifying code (you know, the old trick where you drop the trap code just in front of your feet and step on it)...so double surprise. Triple surprise, when Jim told me on the network that HIS MacII wouldn't run NEON 2.0. Subtle differences in system configuration probably make a big difference here.

The 68020 instruction cache will allow to use self-modifying code, if an address has been executed beforehand that is equal to (modified address modulo cache length), since then the instruction that was modified will always be fetched from memory. If one can make sure that the self-modifying code has stepped through at least 64 32-bit instructions in linear sequence before encountering the modified address, this *might* work. I have traced the NEON code for signs of such a behavior, but no luck. Somehow it works on my MacII and crashes on Jim's...strange and mysterious things going on here.

I bet you'll all like to see a full article on the NEON extensions. As Jim told me, that, or a commercial package, is under consideration, a lot depends on the availability of a more recent NEON release, or even the complete NEON source. The

future of NEON should definitely not end at this point, and any support from outside can only help the issue.

Feedback Dept. Europe

International Conversion
Peter Freund
Sweden

Here's a letter from a reader in Sweden (again those international keyboards and characters...):

"I am one of those non-programming, yet admiring users, that enjoy MacTutor 'cause there is something worth reading there.

I work for Software Plus, distributor in Sweden for several Macintosh and MS-DOS products. We represent products like TOPS, WriteNow, QuarkXPress, SuperPaint, Timbuktu, MacLink and companies like General Computer, SuperMac, Symantec (Living Videotext, Think Technologies), Farallon. I am the one that answers and handles communication products, but also companies "knowledge-base". I also do some translation of programs and manuals.

I know that if an article (like mine) would be published in MacTutor, it would help spreading Macintosh on the international basis and also help our company (I would have less annoying work.) *[Well, there you go! Take your vacation now— JL]*

Take any text from the following letter, in any form, or please write any kind of program that shows how to use the "International Utility Package" (it ought to be quite easy) - so my and all non-english-spoken peoples' job will be easier.

I also keep up a BBS (called Fenix) on a free basis when I am not working (using RedRyder Host). The number is +46 8 308356. (Either I or Fenix answers the phone.) My address is:

Software Plus
Att: Peter Freund
Box 2286
600 02 Norrköping
+46 11 181270
(leave a msg if I am not available)
+46 8 308356
(my usual number ((I work at home with translation of MORE right now)))

I also send some fun [a HyperCard stack for translating] I have created myself. Enjoy!

This is an article I have been wanting to write for a long time: Today 1988-10-05 (US format: 5/10/88...)

How to handle international characters

In the old days, before 8-bit-ASCII, different countries did their own modification to computers, so the national characters would work. In Sweden there even became two versions of 7-bit ASCII, "Swedish usual version" and "Swedish name version".

For example the old Apple II was sold in Sweden with the following character set:

```
 !"#$%&'()*+,-./ 0...9 : ; < = > ? @ A...Y
Z Ä Ö Å ^ _ ` a...y z ä ö å ~ 
```

The ROM was specially modified for Sweden. The keyboard was also modified, in a similar way as a the swedish keyboard nowadays on a Macintosh.

Then came the Macintosh, with RAM-based fonts and complete 8-bit ASCII. Most of the european (all(?) west-european) characters can now be handled in a general way. This means that all well-written programs can be used internationally without modification.

Well, the problems are small compared to our friends in Norway and the IBM PC. When IBM designed their own version of 8-bit-ASCII, they "forgot" the norwegian/danish character Ø. (An error even Apple did with their early version of Courier in Apple LaserWriter...). This means that all ROMs in Norway/Denmark has to be specially made for any MS-DOS machine, and the two characters ¢ and ¥ is replaced by ø and Ø.

There exist more versions of 8-bit-ASCII (Digital Decmulti, Hewlett Packard Roman 8). PostScript use special names for characters. Now let us first look at Macintosh and IBM PC!

When translating between a Macintosh and a IBM PC, it is difficult to only use TOPS or Dynafile for text files. There has to be an international(!) version of MacLink for handling the international characters correctly. With Apple File Exchange for conversion between Apple II and Macintosh I probably obtain the usual [\ instead of ÅÄÖ.

For a very long time there was no program that correctly could use all 256 characters when communicating. Usually the program skipped everything above ASCII 127. Most program have an option nowadays (like Red Ryder 10.3) to turn masking of high bits off.

There is two programs on the Macintosh that handle communication with swedish characters correctly: MacTerminal (swedish version) and inTalk. The user can easily select if swedish (swedish 7-bit-ASCII) is to be used or not. In MacTerminal it is also possible to select if incoming characters, outgoing characters or both should be converted, and even both swedish standards are available.

How to communicate in Swedish?

I guess they use some kind of conversion table. If a Macintosh is to be used as a terminal to a mini in Sweden, there are two conversion tables necessary. Here is the scheme: First I press the swedish character Ä on my Macintosh. The Mac communication program checks if this character is anything that is used for terminal emulation. If not, we convert this character, using conversion table one, into a [. Then we send the character over.

When on the other hand the mini sends a character, we first check if it has anything to do with terminal emulation. Thereafter we convert the characters that is supposed to be sent to screen, using conversion table two.

When MacTerminal, or InTalk implemented the swedish

character set, they did not let the user change the conversion tables. This means that if I want to communicate with a IBM PC and send text files over, I must use some special program again!

Does it really have to be that difficult? In MS-DOS-world there are several programs (Enable, ProComm, plus others), where it is easy to modify a conversion table, with all characters, any way I like. There even exists a TSR (Terminate and Stay Resident)-program (like an INIT) that patches the serial routines for Swedish, so the conversion almost always works.

It ought not to be too difficult to write a cdev that would alter the serial routines, in a similar way and that could be configurable/turned on/off through the Control Panel.

[Such a routine might actually be very useful in a number of circumstances. I can remember the case of Versaterm Pro v.2.0 not recognizing the new ISO Keyboard correctly (the one with the small return key). In that case, you wouldn't be able to use the arrow keys anymore to walk across a VT100 screen. This problem has been solved in never versions of Versaterm, but for the meantime it would have been extremely useful to have a routine available that passes serial in/output through a conversion table like you suggested. If I can figure out how to do such a filter in a way that would satisfy the User Interface Thought Police, I might write a column on it].

To all hackers:

Yet what really would improve things would be if the manufacturers would think a little more internationally. (In Inside Mac there is an "International Utility Package" worth looking at!)

Now, let us look at some popular programs today (I guess most of you know them):

Very Good:

4th Dimension - sorts well, configurable, works even with chinese!

MacTerminal (swedish version).

QUED - sorts and works correctly.

MORE - sorts correctly, handles all characters/ date functions with class. But even the sun has its spots, I cannot use the name "Örjan" as registration name, because it starts with Ö.

Good(?):

WriteNow - can handle foreign characters, but the spelling-checker cannot suggest any foreign characters, and also cannot see the difference between upper case and lower case with international characters.

DiskTop - has a funny bug. Difficult to find by date, because DiskTop suggests international date format, but can only accept US-dates for input. (DiskTop 3.0.2)

Bad (most "famous" programs work pretty well nowadays):

FileMaker Plus - cannot sort correctly. For a DB-program, this is bad. (I think MS File, Reflex and Helix had/has(?) the same problem.)

Old **MacPascal** - not possible to write international characters at all (above 80 hex).

SuperPaint - cannot print on whole "A4 Letter size" (8 1/4" x 11 2/3").

RedRyderHost - uses a font that does not even contain ÅÄÖ.

A lot of programs use Command/] and Command/[and other special characters for fast menu selection, that doesn't work on international Systems.

Finally a suggestion to those "desktop publishing program"-makers that always seems to hunt for good, new features: "Overlay characters"!

Kerning is a modern thing nowadays. With kerning it is possible to combine two characters into one. In many languages it is very usual (as usual as an "e" in english) to combine normal letters (acenorsz) with characters like ˜~^`¨. *[I recently found out to my surprise, when trying to type the address on a letter to Czechoslovakia, that Apple left out a c with an inverted circumflex, as shown below created in MacDraw, from their fonts. In fact, that diacritical mark does not seem to exist].*

Fig. 1 Missing International Symbol

The problem with kerning in desktop publishing programs today is that a value has to be specified. Yet I in most (all?) cases want the top character to be centered above the lower one. I can, with some DTP-programs, put two characters in the same place with kerning, but when size or font changes, the position of the top character goes wrong.."

—Peter Freund

[Thank you, Peter, for these very interesting remarks on international compatibility issues, a specter that has haunted the Macintosh scene ever since this machine was introduced. The fact that changing keyboard layouts 'on the fly' became possible only with System 4.1 has always been embarrassing.

Note to our readers: Peter's letter was accompanied by a file which contained the keyboard layout resources (KCHR) for over ten different countries. This file is included on this month's source code disk for anybody who needs it. -JL]

A Priority Based Task Scheduler

Clive Maynard,
Western Australia

This contribution was sent to us from an Australian Macintosh user and Mach2 programmer. It implements a priority-based scheduling system for Mach2 tasks.

"Dear Jörg,

I have read your column in MacTutor for quite a time with

considerable interest and have found it very useful.

I have enclosed on disk something which may be of interest to you for an article in MacTutor.

The software provides a demonstration priority based scheduler for Mach2.

The basic round robin is not removed for the Mach2 task and Editor together with a background scheduling task. Additional tasks are added in such a way that they put themselves to sleep on completion through the execution of a word SwitchTask and return control to the scheduler. The scheduler determines from the priorities of available tasks which should be woken next, puts itself to sleep and wakes up the correct task.

The result is that the minimum number of tasks are in the round robin when the normal Mach2 pause occurs.

Variations which could improve speed include coding the scheduling task in assembler but this was developed for student study of operating system functions not high performance.

Priorities may be changed through a Priority Terminal task window in the range 0 to 100. 0 corresponds to task suspension until the priority is raised back to a positive value. The initial default priority is 10 for each task.

[As you can see from the code, the scheduler creates a task table for a maximum of ten tasks. For each task, its current level, the priority value and its address are maintained in this table. The level is initialized to zero. In its main loop, the task scheduler decrements the level of each task by its priority value. Each time the decrement passes through zero, the task is woken up for one round of execution and the level reset to 100. —JL]

The three additional tasks run in the demonstration provide a simple quickdraw output. The tasks can have their priorities varied and easily show the effects of the priority allocation.

The only other features which may be of interest are:

1. The creation of a new word LVALLOT which allows for the creation of local arrays in words with local parameter and variable facilities.

2. #IN which is an intrinsic word in UR/FORTH for the IBM/PC is created here using LVALLOT and the SwitchTask function to prevent the suspension of the other tasks when new priority values are being entered.

Clive Maynard,
Wave=onic Associates
199 Watts Road,
Wilson 6107
Western Australia"

[One further remark is that one should minimize the use of PAUSE in this task-switching environment, and use SwitchTask instead to make the scheduling more effective. For this purpose, the word PTExpect redefines EXPECT to contain a SwitchTask. There are PAUSEs left (in ?TERMINAL and KEY), but this can't be avoided.

The illustration shows the nice graphical output of Clive's demo. —JL]

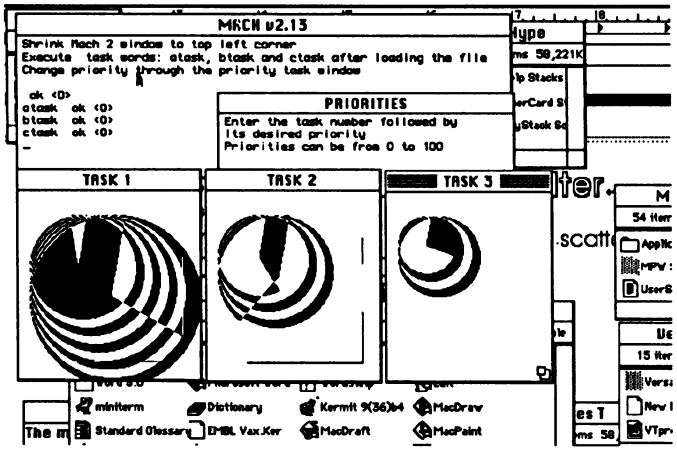


Fig. 2 Mach2 task scheduler output (on JL's usual crowded Multifinder desktop)

The Mach2 task scheduler

```
.( Priority based multitasking on the Macintosh ) cr
( Original concept: D.Bryant, G.Caunt, G.Else 1987 )
( Modifications and Generalisation: )
( C.A.Maynard 1988 Wave=onic Associates)
( Version 1.1 060488 )
Decimal
( ------------------------------ )
( Task and window configurations for the necessary tasks )
( ------------------------------ )
400 1000 background schedulertask
400 1000 terminal prioritytask

new.window PriorityWindow
" PRIORITIES" PriorityWindow Title ( create priority window )
40 250 110 500 PriorityWindow Bounds
Document Visible NoCloseBox GrowBox PriorityWindow Items
PriorityWindow Add
( ------------------------------ )
( DEMO Tasks and windows )
( ------------------------------ )
400 1000 terminal task1
400 1000 terminal task2
400 1000 terminal task3

new.window Task1Window
" TASK 1" Task1Window Title ( create the TASK 1 window )
140 20 310 180 Task1Window Bounds
Document Visible NoCloseBox NoGrowBox Task1Window Items
Task1Window Add

new.window Task2Window
" TASK 2" Task2Window Title ( create the TASK 2 window )
140 180 310 340 Task2Window Bounds
Document Visible NoCloseBox NoGrowBox Task2Window Items
Task2Window Add

new.window Task3Window
" TASK 3" Task3Window Title ( create the TASK 3 window )
140 340 310 500 Task3Window Bounds
Document Visible NoCloseBox NoGrowBox Task3Window Items
Task3Window Add
( ------------------------------ )
( USER Variable definitions )
( ------------------------------ )
72 user TaskWindow
220 user rleft
224 user rright
```


© The Definitive MacTutor, Vol. 4

```
228 user rtop
232 user rbot
236 user diff
      ( create rectangle coords as user variables for each task
)
240 user angle
244 user rectangle  ( space for 8 bytes needed. Next slot 252)
( ------------------------------ )
( Scheduling Task Definitions )
( ------------------------------ )
VARIABLE LevelAddr
         ( Global temporary storage for the scheduler )
VARIABLE NTASKS  ( Number of runnable tasks  MAX 10)
0 NTASKS !          ( Initialise to zero )
VARIABLE PTASKS 120 VALLOT
            ( Storage for Level, Priority and Task Address )
: Wakeup ( a1 - )
( Wakeup gets the next task running given the status address )
sleep status W!
wake swap W! pause ;

: SwitchTask
( Call the scheduler to see who's next )
sleep status W!
wake status task-> schedulertask W!
pause ;

: SCHED
( Define a general task scheduling process )
activate.
begin
NTasks @ dup 0> if  ( only execute defined tasks )
0 DO
   I 12 * PTasks + dup LevelAddr ! @
                ( Get the address and current level )
   LevelAddr @ 4 + @          ( Get priority setting )
   + dup LevelAddr @ !        ( Save new level )
   100 - dup 0> if                                  ( Modify
level if necessary )
      LevelAddr @ !
      LevelAddr @ 8 + @ Wakeup
   else
      drop
   then
LOOP
else drop then pause
again ;
( ------------------------------ )
cr
.( Clive Maynard's Forth Environment extract ) cr
.( C.A.Maynard 020488 ) cr

also assembler

code LVALLOT ( n - addr )
( Set up a local buffer. )
( Only callable from a word with local variables )
( UNLK will clean up the stack. )
( USER beware of buffer overflow!!! )
   MOVE.L (A6)+,D0  ( GET SIZE IN BYTES )
   MOVEA.L   (A7)+,A0  ( GET RETURN INFO )
   SUBA.L D0,A7     ( NEW SP )
   MOVE.L A7,-(A6)
            ( COPY ADDRESS TO PARAMETER STACK )
   JMP (A0)
end-code

-1 CONSTANT TRUE
0 CONSTANT FALSE

: #terminator? ( char - f )
( check for the terminator of a number: space or CR )
case
13 of true swap endof
32 of true swap endof
false swap
endcase ;

: PTexpect ( buffad nchars | buffadd countup - )
( Fills a buffer but includes Priority Task switching )
buffad 1 + -> buffadd 0 -> countup
nchars 0 do
   begin
   SwitchTask
   ?terminal until
   key dup emit dup buffadd C! 1 +> buffadd 1 +> countup
   #terminator? if leave then
loop
countup buffad c! ;

: #IN ( | buffaddr - number )
( PC/FORTH intrinsic function!! )
( Collect into a 10 byte buffer and return a number input )
10 lvallot ( set up a local buffer very carefully )
 -> buffaddr
buffaddr 10 PTexpect
buffaddr number? drop ;
( ------------------------------ )
( A new task building word for Priority Tasks )
( ------------------------------ )
: PBUILD ( TaskAddr | LevelAdr - )
( Initialise conditions for new tasks )
NTasks 10 = abort" Task Priority Table Full. New entry denied"
NTasks @ 12 * PTasks +              ( Get offset into table )
dup -> LevelAdr 0 swap !            ( Set Level to zero )
10 LevelAdr 4 + !                      ( Set Priority to de-
fault of 10 )
TaskAddr BUILD                            ( Now do an ordinary
task build )
TaskAddr @ LevelAdr 8 + !        ( Save Status address )
NTasks @ 1+ NTasks !                ( Increase task count )
;
( ------------------------------ )
( The Priority Task Definition )
( ------------------------------ )
: getbuff ( | taskno priority - )
( priority reallocation routine)
#in dup ." Task " . -> taskno cr
taskno NTasks @ < taskno 0> and if
#in dup 0< if drop 0 then dup ." Priority " . -> priority
      ( New priority determined )
   priority 101 < IF
      priority dup 0= if
         cr ." Zero or negative priority halts the task" cr
          then
   taskno 12 * 4 + PTasks + ! ( Get to storage location )
   ELSE cr ." Priority out of range. No change" cr THEN
else cr ." Task number out of range. No change" cr then ;

: setpriority ( priority allocation task)
activate
begin
taskwindow @ call SetPort
." Enter the task number followed by" cr ." its desired priority"
cr
." Priorities can be from 0 to 100" cr
   ?terminal if
   getbuff cr then
   SwitchTask
again ;
( ------------------------------ )
( The other tasks )
( ------------------------------ )
: DISKS
activate
                              ( Initialise variables )
```

```
-7 diff ! 0 angle !
10 rleft W! 20 rtop W!
150 rright W! 160 rbot W!
( Loop through graphic changes in superb animation )
begin
   rectangle rleft W@ rtop W@ rright W@ rbot W@ call SetRect
   begin
      359 angle @ - 0>
   while
      taskwindow @ call SetPort
      angle @ 10 + angle !
      rectangle angle @ 10 call InvertArc
      SwitchTask
   repeat
   taskwindow @ call SetPort
   rright W@ diff @ + rright W!
   rbot W@ diff @ + rbot W!
   rright W@ rleft W@ - 5 < if
      7 diff ! then
   rright W@ rleft W@ - 140 > if
      rectangle 0 360 call EraseRect -7 diff ! then
   0 angle !
   SwitchTask          ( Get back to the scheduler )
again ;
```

```
( ------------------------------ )
( Initiate the necessary tasking operations )
( ------------------------------ )
schedulertask build ( slot scheduler into round robin loop )
schedulertask sched
prioritywindow prioritytask Pbuild
prioritytask setpriority
( ------------------------------ )
.( Define task insertion words to show adding tasks to priority
system ) cr
.( Shrink Mach 2 window to top left corner ) cr
.( Execute  task words: atask, btask and ctask after loading the
file ) cr
.( Change priority through the priority task window ) cr
( ------------------------------ )
: Atask task1window task1 Pbuild
task1 disks ;

: Btask task2window task2 Pbuild
task2 disks ;

: Ctask task3window task3 Pbuild
task3 disks ;
```

Jörg Langowski
Wayne Joerding

MacTutor Vol. 4 No. 8

Object-Oriented Extension to MACH2™ Forth
by Wayne Joerding

[This is the first of - hopefully - several upcoming articles dealing with Object Forth extensions. I introduced you to some of the features of Wayne's implementation in my last column. Now, here comes the real stuff. I checked out Wayne's code several times and it seems to work very well. For those of you who play with the source code disk before reading the article I should point out that you have to turn vocabulary hashing OFF before you try out the code; this is also explained in the article.

Wayne Joerding is an associate professor of economics at Washington State University in Pullman, WA. He uses the Mac - with Mach2 and his object extensions - in his research on stock market efficiency issues. He can be reached on GEnie - W.JOERDING.

Wayne went through great lengths in the implementation to make sure that the code works with WORKSPACE, NEW-SEGMENT, and TURNKEY. Because of the way dictionary links are changed by his code during class compilation, this is not at all trivial. Read his notes carefully under this aspect. - JL]

Introduction

How could any programmer resist "object-oriented programming," with its promise to reduce program development time through reusable code and improve revisability by the use of self-contained modules that encapsulate data structures and their procedures that operate on the structure. Then to find out that the whole thing works by passing messages to something called an object, it was too much to resist. But, I didn't want to give up my experience with Forth, its fast code, extensible compiler and interactive programming style. Fortunately, it is possible and desirable to build an object-oriented extension of the Forth language, NEON is perhaps the best known example.

Brad Cox's excellent book *Object-oriented Programming: An Evolutionary Approach*, promotes the advantages of using a hybrid language derived by extending a traditional language to accommodate the object-oriented approach. *Object-oriented Forth* by Dick Pountain actually shows how to add abstract data types (objects) to Forth. Armed with these two books, I have developed an object-oriented extension to MACH2 Forth. This article describes the results. My approach is not the only one, NEON has already been mentioned and there are postings on GEnie from others who are making object-oriented extensions to FORTH. However, each approach has unique features and limitations, thus an additional objective for this article is to discuss some of the problems and tradeoffs I faced in the process. Here is an abbreviated version of a summary sheet like that used by Kurt Schmucker in *Object-oriented Programming for the Macintosh.*

ObjectFORTH	
Background Information	
Programming Environment :	MACH2 FORTH
Toolbox Access :	Yes
Object-Oriented Information	
Instance Variables and Methods :	Yes
Class Variables and Methods :	Yes
Multiple Inheritance :	No
Unique Instance Methods :	No
Number of Classes in library :	1
MacApp access :	No
Dynamic Objects :	No

First, let's clarify some jargon used in the literature. Objects are an abstract construct composed of a data structure and associated procedures. Objects are organized into child-classes of a root class, each child-class having more specifically defined methods and data structures than its parent-class. For example, "stacks" are a child-class of "ordered lists" which are a child-class of "arrays". Each class is composed of instances of the class. For example, a particular stack can be an instance of the class "stack". An object's data structure consists of named fields called instance variables. The procedures are called methods and are like subroutines. A program requests an object to execute a method by sending the object a message which is used by a "selector" routine to select the appropriate method. The set of messages to which an object responds is called a protocol.

There are two essential features of object-oriented programming, encapsulation and inheritance. Encapsulation refers to the isolation of an object's data and methods from other parts of the program. Encapsulation is enforced by restricting access to an object's data or procedures to a standardized message interface used by each object. Objects are presented a message to which they respond by selecting a method to execute. The specific method is an internal matter for the object and depends on the implementation. For example, a link-list can be implemented as an ordered array or an unordered array with pointers. The importance of encapsulation is that a user of an object does not need to know how the object is implemented. In a sense, encapsulation is nothing more than a strong form of good modular programming practice. However, object-oriented programming not only enforces modular code but also bundles the data structures with the procedures.

Inheritance refers to the ability of objects in a child-class to inherit the structure and procedures of the parent-class. Furthermore, objects in the same class, called instance objects, share the same code. The advantage of inheritance is that data structures and procedures can be reused and extended.

In addition to meeting the above requirements, I also had several design goals for the extension. The most important

influence on the extension described here is its pedagogical nature. I was trying to learn about object-oriented programming from the extensions and so I made the logical and physical data structures as similar as possible. The logical structure adopted is that given in Cox's book and is a natural choice for a FORTH programmer because of its reliance on linked lists.

A second design goal was to enforce a strong degree of encapsulation by requiring messages to be simple character strings, without any procedural capability of their own, and by hiding all forth words which identified methods or instance variables. Thus, trying to execute the name of a method or instance variable will only get you a beep and a question mark. An alternate approach is to have method names put an identifier on the stack and then have the object use that identifier to find the appropriate method. A disadvantage in this approach is that forgetting to use the object does not immediately result in an error. A disadvantage of my approach is that it requires rather major surgery on the links of the dictionary, a risky process which slows compilation.

A third design goal was that every construct in the extension should be an object in its own right, each object searching the input stream for a message. This means that the creation of subclasses is a method of a parent-class object. As a consequence, objects are divided into two groups, "class defining objects", CDO's, and "instance objects", IO's. Instance objects are members of a class that is defined by a class defining object. CDO's are the factory objects of Brad Cox, their main purposes are (i) specify the data structure and methods that define the class and are used by instance objects, (ii) make instance objects, and (iii) create new child-classes. I will discuss each of these capabilities in turn.

The data structure and methods of an object are implemented as normal forth words which are sealed from the rest of the forth dictionary by redirecting the links of their headers, more on this later. Access to the data structure and method names is controlled by the class defining object. A CDO gives a new instance object a pointer to the link list of instance methods and instance variables that define the class.

Only CDO's can make instances of a class. Each instance can respond to a protocol of messages as specified by its class definition. For example, if Forth's "Pstack", for parameter stack, and "Rstack", for return stack, were defined in an object-oriented language then they would both be instances of the class "stack", and as such would respond to the message "pop" because that is a message in the protocol of all instances of "stack". The instance " Pstack" would contain all the characteristics peculiar to the parameter stack, such as data, while the instance "Rstack" would do the same for the return stack. The CDO "stack" defines the structure of the data contained in the instances "Pstack" and "Rstack" and the methods to which these instances respond.

Only CDO's can create new classes. It is the creation of child-classes that implements the inheritance of methods. A child class inherits the methods and data structure of its parent-class. For example, "stack" is a child-class of the class "list", thus "list" passes along its data structure and methods to its child-class "stack". Consequently, if instances of "list" respond to the message "size", instances of the class "stack" also respond to "size". If the programmer wants instances of the class "stack" to respond to "size" in a different manner than instances of the parent-class "list", then the "size" method must be overridden by specifying a new method for the message "size". This sounds great in theory, but practice reveals one of life's tradeoffs. Take our example, one could code a "size" method which works ineffieciently for all possible child classes, or one could code a method which is optimized for a "list" type object and later override the "size" method with one which is optimized for a stack type object. The class creation procedure also provides a way to add additional methods to the child class, such as "pop" for the "stack" class.

There are important advantages to requiring all constructs to be objects. An obvious advantage is the uniform handling of objects. Making instances and subclasses is just the same as accessing data in an instance object. A much more important advantage is that instance and child-class creation are methods, methods which can be be overridden by child-classes. In a previous example it was pointed out that the "size" method of a parent-class can be overridden by the child-class, similarly, the instance and child-class creation methods can be overridden. This ability to override the methods of a class defining object is analogous to the extensibility of the forth compiler, and has the same importance. One of the code examples shows the use of this feature to alter the instance creation method so that it forms instances of variable size. Another example uses this feature to implement a C-style data structure.

A fourth design goal was to make the compiled code as fast as possible. Slowness has been a criticized characteristic of object oriented languages, although the speed sacrifice can be quite small (see Cox's book for a discussion and references regarding this issue). The desire for speed led to a binding scheme like that used by Pountain. Binding refers to the point in time when a label is associated with a value. For example, the definition " : myDUP ['] DUP EXECUTE ; " would bind early the CFA of DUP to the label myDUP. On the other hand, the definition " : PLEASE ' EXECUTE ; " and used as PLEASE DUP would be an example of late binding of the label PLEASE to the CFA of DUP. My extension uses early binding in the sense that methods are chosen at compilation time rather than run time. It uses late binding in the sense that variable names are associated with addresses at run time. The essential overhead involved with this approach is that a reference to any object instance variable in a method definition requires a number to be read off a special object stack and added to an offset number on the regular FORTH parameter stack. Any implementation of a data structure as complex as an array would require adding some offset value, so the extra fetch from the object stack is the actual overhead. Please see Pountain's book for a discussion on the whether this overhead is worthwhile.

Finally, a fourth design goal was to make the extension itself use abstract data structures in order for the code to be as easy to change and maintain as possible. Thus, the data structures in class defining objects are accessed by their names as much as possible, not by calculating an offset which requires unchanging knowledge of the format of the data structure layout. The only fixed address is the field used by a CDO to store the LFA of its first method, this variable must be in the first long word of a CDO's data set. A consequence of this goal is that the code sometimes uses the trick of temporarily pushing an object's address on the object stack and then popping it off when no longer needed.

Syntax and Usage

This section describes the syntax used to send an object a message, create a new class, and make an instance of a class. Additionally, I describe some of the built-in methods and tips on using ObjectFORTH.

Send any object a message:

```
Object.1 Msg
```

Messages cannot be larger than 32 characters. Capitalization matters! These restrictions are due to the simplicity of the selector routine used to find methods.

Make an instance with the identifier INS.1:

```
CLASS.1  Make.Instance  INS.1
```

Make.Instance is a method of the class defining object CLASS.1, it simply uses the character string INS.1 from the input stream to create a dictionary header for the new instance.

Create new class with the identifier CLASS.2:

```
CLASS.1 Define.Child.Class
    :Class
        I.Var <name.1>
    Hide
        :M <name.2> <normal forth def> ;
    ;Class
        k Constant <con.1>
        Variable <var.1>
    :Instance
        Variable  <var.2>
        I.Var <name.3>
        n Ins.Array <name.4>
    Hide
        :M <name.5> <normal forth def> ;
    ;Instance
CLASS.1 Name.Child.Class  CLASS.2
```

Each CDO has two sections, a class section which contains the methods and variables of the object and an instance section which holds the methods and data structure for instances of the class. The words :Class and ;Class start and end definition of the class section of a child class, similarly for :Instance and ;Instance. Each of these parts of the definition are optional. Thus, most CDO's have no need to add methods to the class section of the object. The order of the class and instance section definitions does not matter.

The word I.Var creates a long word integer variable which is inherited by each child. That is, an I.Var in the class section is inherited by any CDO which has CLASS.2 as an ancestor, and an I.Var in the instance section is inherited by each instance of the class defined by CLASS.2 and by each instance that has CLASS.2 as an ancestor. The word Ins.Array is similar but creates an array of n bytes. :M starts the definition of an object method. In this example, :M defines a new class method named <name.2>. The method <name.2> is used by sending the message <name.2> to the object CLASS.2.

The word Hide prevents previous words in a section from being available as methods. For example, use of Hide in the class section prevents the <name.1> instance variable from being available as a method name. In this way the user of an object cannot gain direct access to the memory location of <name.1> by sending <name.1> as a message. Hide is usually used in this way, to hide an object's data structure, but it can also hide methods. Thus, one could define a method before Hide and it would not be available as a public method, only internally to the object. Hide is optional.

Note that normal forth constructs, such as Constant, can be used in the definition. How these constructs are used depends on their placement. If placed outside the section defining words (:Class and ;Class or :Instance and ;Instance), such as <con.1>, then the word is not available in any fashion after the object definition is completed. But the word can be used in method definitions. This is a good way to define words used in method definitions but which are not subsequently desired as methods. This should be contrasted with the effect of Hide discussed above. For example, the variables <var.1> and <var.2> both provide a" class variable" for CLASS.2. (Class variables are variables shared by instance objects of the same class, which can be used for passing data between instances.) The difference between the two is that <var.1> is not available to descendents of CLASS.2, while <var.2> would be available to new methods of CLASS.2 descendents. That is, methods defined in children of CLASS.2 could use <var.2> as a variable. However, <var.2> is not passed on to a child class in the manner of an instance variable. If forth definitions are placed inside the section defining words but after Hide then the construct is available as a method by its name.

There is an additional method included in the root object besides child-class and instance defining methods. This method is Describe which describes the an object. This method works for both CDOs and instances, but produces somewhat different results in each case. In the CDO case Describe prints the name of the object, some information about the object, then sends itself the messages Pr.ob.ivar, Pr.class.meths, and Pr.ins.meths. These methods list the ivars, class methods, and instance methods, however, only Pr.class.meths and Pr.ins.meths will work for child objects. Pr.ob.ivar will not work for a child object which adds ivars. This example demonstrates one of the subtle features of programing with objects. It takes extra thought, and sometimes is impossible, to define a method that works properly with child classes.

There are several important features of Object FORTH that affect usage. First, the data hiding and encapsulation are accomplished by redirecting links in the LFA of a name header. This hiding will be defeated and inheritance will not be implemented properly if HASHING is used. Thus, you must turn off the HASHING feature of MACH2. Second, only those methods which have been declared GLOBAL can be used outside a code segment, even interactively. In this regard we are very fortunate in the way MACH2 implements NEW-SEGMENT. MACH2 maintains a separate link list of words which are to have jump table entries, then during execution of NEW-SEGMENT this link list is traversed, changing the Segment Field of the dictionary header to the appropriate number and putting the jump table offset into the Parameter Field Pointer region of the dictionary header. Thus, those words designated as GLOBAL will have a correctly defined dictionary header after NEW-SEGMENT. However, the other methods have been delinked from the normal search order, so the dictionary header of these words is not found and changed by NEW-SEGMENT. What is needed is a new

NEW-SEGMENT, a task which remains for the future. Third, you must execute Init.Ostack each time you start Object.FORTH from a file which was created by NEW-SEGMENT. Also, you must make Init.Ostack part of the initialization procedure in a TURNKEY'ed application. The reason for this is that the object stack, to be described later, is based on the variable Ostack and must be initialized to the proper value before use by objects. You have been warned, this one will sting you at least once!

Those familiar with other object oriented languages may have noted the absence of SELF and SUPER. These pseudo objects are avoided here in preference for the natural Forth ability to call any previously defined word. If you're not familiar with these SELF and SUPER then ignore this paragraph.

Implementation

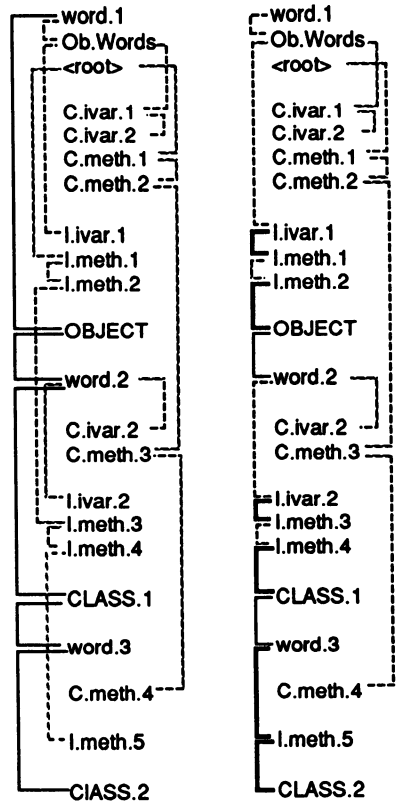
I will describe the implementation by sequentially discussing the most important words in the accompanying code. But first, it is best to describe the general structure of the implementation. Figure 1 shows the links between words before and during the definition of the instance part of a class defining object. It is supposed that the class part of the root object OBJECT has instance variables C.ivar.1, C.ivar.2, and methods C.meth.1, and C.meth.2, while the instance part has variables I.ivar.1 and methods I.meth.1 and I.meth.2. CLASS.1 is a child class of OBJECT, it has a class part with variable C.ivar.2 and method C.meth.3, it has an instance part with variable I.ivar.2 and methods I.meth.3 and I.ivar.4. CLASS.2 is a child class of CLASS.1, it only has one class method of C.meth.4 and one instance method of I.meth.5, no variables. The words word.1, word.2 and word.3 are normal forth definitions.

The left side shows the links before creation of a new child class begins. The solid lines represent the links followed by the forth interpreter, thus an execution of WORDS would result in CLASS.2 word.3 CLASS.1 word.2 OBJECT word.1 etc. The dashed lines represent internal links of an object. Data fields in the class defining object control access to these link lists. For example, an instance of CLASS.1 would use the pointer in a field named ins.Key to find the head node for its link list of methods, at I.meth.4. This link list of methods would then be traversed by a routine called Selector searching for the appropriate method. If the search got as far as <root>, whose link is zero, then the method is not found and an error message is reported. Similarly, a field named class.Key points to the head node of methods (i.e. C.meth.3) for the class defining object CLASS.1. Inheritance is implemented by linking methods of child classes to the methods of parent classes. It is also easy to see the effect of Hide. In the example all the instance variables are hidden by directing the link of the last method of each object to the first method of its parent object. That way Selector won't find the method or instance variable name.

The right side shows the links during the definition of the instance part of the new class defining object. Two types of changes have occurred. First, Define.Child.Class makes a change which is not shown, it relinks the section defining words :Class, ;Class, :Instance, ;Instance, and Hide, so that they can be used during the class definition. Second, :Instance changes the links shown in heavy lines. For each ancestor object there are two link changes which must be made. First the object name is linked to the head instance method, for example, CLASS.1 in linked to I.meth.4. Second, the tail instance method is linked to either (i) the next public word (as with I.meth.5 to word.3) or (ii) the first hidden word (as with I.meth.3 to I.ivar.2). The difference occurs between those objects that do not have a hidden part, such as CLASS.2, and objects that do have a hidden part, such as CLASS.1. In the case of CLASS.1 the last hidden word is already linked to the next public word (i.e. I.ivar.2 is linked to word.2). In the case of CLASS.2 there is no hidden words so the last unhidden method is linked to the next public word (i.e. I.meth.5 is linked to word.3). For the root object the next public word is defined to be Ob.Words. At the conclusion of defining the instance part ;Instance will return the links to their previous condition. Then it only remains for Name.Child.Class to delink the words :Class, ;Class, . . . Hide.

The memory location for each type of object is quite different, but each object has a data structure starting at an address stored with the name of the object. I call this address the data field address, DFA. Thus, the DFA is the address where the object's data structure starts. The data structure of a CDO is stored along with the dictionary headers, while the data structure of instance objects is stored in the Forth code segment by a standard ALLOT statement. The reason for this difference is that the data for a class defining object is only needed during compilation and can be disposed when compilation is complete. This is automatically done by TURNKEY. The name of a class defining object only stores the handle (not a pointer) to its data,

Figure 1



whereas the name of an instance object stores the actual instance data. The only physical constraint on the data of a CDO is that the instance variable class.Key (which contains a relative address pointer to the first class method) must be at a zero offset from the value pointed to by the contents of the handle.

Now let's look at the code. First comes the memory allocation for the object stack, enough to accommodate nested objects to 20 levels. Ob.DefWords and ObjectFORTH are markers which are used by Define.Child.Class to relink and delink the words in between. The section defining words :Class etc. are deferred so that their definitions can use the data structure names defined for the root object. Next come global variables used in the construction of new objects. Then the object stack manipulation words are defined. Note that the push and pop words contain one line of code for error checking which can be removed after debugging. Ins.Array, I.Var, and I.Pntr are defining words used to define an objects data structure. Let's use PFA as one would for an indirect threaded Forth, i.e. PFA is the address put on the stack before execution of the DOES> code. Ins.Array creates an immediate word that takes an offset value from its PFA, compiles it as a literal, then compiles a JSR to Ocop+. The created word is the name of a variable or array. Ins.Array keeps track of the offset to the data for each new variable name in the variable Offset, which is incremented by the value 'size'. I.Var is a special case of Ins.Array. I.Pntr is a special defining word that doesn't increase the offset value, which is useful in constructing variable sized objects, but must be used with care. I.Pntr cannot be followed by Ins.Array or I.Var or the resulting instance variables will not have the correct offset.

The method defining word, :M, is nothing more than the usual colon definition, but changes could be made in the future. For example, :M could look for a duplication of the new method and issue a warning.

A special abort routine is defined. This abort routine is turned on by Define.Child.Class and turned off by Name.Child.Class. The reason for a special abort routine is that a compilation error while a child class is being defined leaves the hidden part of the object relinked to the forth dictionary. Unfortunately, because of the way :Class and :Instance work a subsequent attempt to compile the class definition, and consequent execution of :Class or :Instance, causes the dictionary to be relinked in such a way that the first method of the nearest ancestor class is the first word searched by the forth interpreter. This leads the interpreter to <root> which has a zero link, terminating search of the context vocabulary and leaving the user shut out from the default Forth words.

The next group of words are used in connecting messages with methods. The most important word in this section is Selector. The 'key' input argument to Selector is the LFA of the first method of an object. Selector then gets the next character string in the input stream and uses it to identify the appropriate method. Get.Meth finds the method and passes the methods CFA to Do.OR.Compile, which executes the method if the state is interpret and compiles the word otherwise. The best way to understand these words is to first look at Do.Meth. Do.Meth pushes the object's data field address, DFA, on the object stack, then execute the method's CFA, and finally pops the object stack. Compile.Meth compiles code which does the same thing at run time, except that it compiles a JSR to the method.

The actual definition of OBJECT begins with Ob.Words,which is used by :Class and :Instance to relink the previous words so they are visible by the Forth interpreter. <root> acts as a null CDO or a stopper for method link lists by setting its LFA to zero. <root> acts like a CDO by having a handle to data in the name area that is set to zero. The variable #c.Pvt.Tail is initialized to what eventually will be the CFA of the last word in the hidden class part of OBJECT. Offset is initialized to zero. All of the instance variables used by OBJECT are then defined and Offset is saved in #class.Size for later storage in OBJECT's data structure as class.Size (Remember that Offset is used by the variable defining words, such as I.Var to accumulate the size of the instance object's data set..

The next four words are used in class methods but are not available as methods. SetClassStruc, the most important, initializes the data structure of child classes. It is defined here so that it can use the variable names defined by the previous I.Var definitions. In order for the link changes to work after execution of NEW-SEGMENT all LFA's must be converted to a relative link address, RLA, from a dictionary base pointer. A call to ClassStrucAllot with argument #class.Size gets the necessary space in the name area. ClassStrucAllot is set up to leave an error message but this feature is not implemented yet. After the space is allocated the returned address is duplicated. One copy is converted to a relative address and stored in the next memory location as a handle, the other copy is pushed onto the object stack. With the new object's DFA on the object stack the instance variable names can be used to store the needed data. For example, the LFA of the head method for the class defining object is fetched from the variable #c.Head, converted to a RLA and stored in the field called class.Key. Further explanation of SetClassStruc is best done along with a discussion of the Define.Child.Class and Name.Child.Class, below. Next, the variable #c.Pub.Tail is initialized. This mimics the action of Hide.

The words Pr.ob.ivar, Pr.class.meths, Pr.ins.meths, and Describe are methods that can be used to describe a class defining object. As discussed earlier, Pr.ob.ivar only works with OBJECT, it won't print any new instance variables in a child class.

The next three methods are the most important. The first two control the creation of child classes and the third makes instances of the class. Their use was described in the syntax section. Define.Child.Class initializes temporary variables used during the definition of a child class. Name.Child.Class actually creates a new child class, it sets up the data structure, seals the object so that the Forth interpreter cannot find its methods or variable names, links the tail method (which is the first method defined for a child class and the last one in the search order) of each part (class or instance) of the object to the head method (first in the search order) of the parent class and delinks the section defining words. Lets examine these methods, along with SetClassStruc and the as yet undefined words :Class and ;Class. The words :Instance and ;Instance behave similar to :Class and ;Class, so are not discussed.

Please examine each step in Define.Child.Class, in most cases the value contained in an instance variable of the object executing the Define.Child.Class method is stored in a temporary variable. In three cases an RLA is converted to a LFA. The LFA of the most recently defined public forth word is saved in #Forth.Head for use by the SetClassStruc routine. Saving

class.Size, ins.Size, class.Key and ins.Key implements the inheritance feature of object oriented programming. The variables #c.Pvt.Tail thru #i.Pub.Tail are set to zero to flag the use or not of the section defining words :Class and :Instance. The variable #parent is set to the relative DFA of the object executing Define.Child.Class. The words :Class etc. are relinked to the dictionary and the custom abort routine is installed. This completes the setup, we're now ready to execute :Class or :Instance.

:Class has two types of actions. First, it initializes certain variables. The size of the class defining object's data structure is obtained from #class.Size and stored in Offset, remember that words like I.Var use the variable Offset. The variables #c.Pvt.Tail and #c.Pub.Tail are initialized to the same value. Each holds the future CFA of the tail word in the class section of the new object. This is converted to an LFA later. The variable C.or.I? is set to flag that compilation is inside the class section of the new object. The second type of action is to relink all ancestor class methods to the dictionary by executing Relink.Parents. Note that :Class is not a method, so it doesn't have access to the instance variables of the object which executed Define.Child.Class.

Relink.Parents is one of the most difficult words in the entire code. It begins by pushing the DFA of the object executing Define.Child.Class (stored in #parent by Define.Child.Class) on the object stack in order to use instance variable names in the definition. Next, it checks the C.or.I? flag and then branches to the correct loop. The Begin-Repeat loop relinks each ancestor object and exits when it reaches an ancestor object with a class.Key of zero, an event which occurs when relinking reaches the null object <root>. The following is executed for each ancestor object, say Ob(i). If class.Key is not zero it checks class.Tail to see if Ob(i) has a class section. The instance variable class.Tail contains the value of #c.Pub.Tail at the time SetClassStruc was executed during the definition of Ob(i). Remember, #c.Pub.Tail is set to zero by Define.Child.Class and it is changed to non-zero by :Class. Thus, class.Tail is zero if the Ob(i) has no class section, and is non-zero if Ob(i) has a class section.

If there is no class section of Ob(i) then no relinking should be done. In this case Relink is skipped, the next object's DFA is calculated from the contents of ivar Parent, the old DFA is popped from the object stack, and the new DFA is pushed on the object stack. Thus, when the loop starts again with a fetch of class.Key it will operate on the parent of Ob(i). In the case that class.Tail is not zero then relinking must be done. The current value of the objects LFA is stored in the ivar Ob.name.link for future restoration. Then, the offset to the head class method is stored in the LFA of Ob(i). Finally, the contents of the tail method's LFA are swapped with the value in c.Tail.link.

Hide can be used during the definition of a class or instance section. Hide simply resets# c.Pub.Tail, if defining a class section, or #i.Pub.Tail, otherwise. This makes #c.Pvt.Tail and #c.Pub.Tail different, so that later one can tell if Hide was used.

The word ;Class has two parts. The first part stores the LFA of the head class method in the variable #c.Head, saves Offset in #class.Size, and converts the CFA of the words pointed to by #c.Pvt.Tail and #c.Pub.Tail to LFA's. Remember that these two values are different if the word Hide was used to hide some of the methods or instance variables. The second task of ;Class is to delink the ancestor objects. This process is essentially the inverse of the relink process except that the contents of class.Key are not needed, because the original contents of the objects LFA were stored in Ob.name.link.

Name.Child.Class actually sets up the new object. First, it delinks the words :Class etc., restores the normal abort routine, creates an immediate name for the new class defining object, and then enters SetClassStruc. The SetClassStruc routine has been partially explained. Now let's examine the handling of c.Tail.link and i.Tail.link, which, as we have seen, each store the required link change for the tail unhidden method of each section of the class defining object. Let's consider c.Tail.link as an example. Recall that only two links need to be changed by :Class. The second link change requires that the tail unhidden method be redirected to the first hidden word, if there is one, or the next regular public Forth word, otherwise. Which specific change to make is determined at this point and stored in the field c.Tail.link for later use. The variables #c.Pub.Tail and #c.Pvt.Tail contain the LFA of the tail unhidden method and tail hidden method, respectively. If there is a hidden part then the contents of #c.Pub.Tail and #c.Pvt.Tail will not be the same, (because of the action of Hide.) so c.Tail.link should be set to the default contents of the tail unhidden word's LFA, i.e. to the contents of the address pointed to by the contents of #c.Pub.Tail. If the contents of #c.Pub.Tail and #c.Pvt.Tail are the same then there is no hidden part (Hide has not been used) and the offset to the next public word is calculated (using #Forth.Head) and stored in c.Tail.link. Finally, notice that if the new object has no class section (specifically, if ;Class was not executed.) then #c.Head contains the LFA of the head class method of the parent class, which is stored in the class.Key field. Similarly for #i.Head and ins.Key.

The final task of Name.Child.Class is to seal the object and link it to the parent object. Alteration of the links occurs in the two IF-THEN parts of Name.Child.Class. If there are no additions to methods or instance variables in the class part of the new CDO then no link changes should be made. Determining whether there has been additions or not is made by examining #c.Pub.Tail (#c.Pvt.Tail would work just as well), which was set to zero by Define.Child.Class and would only be non-zero if :Class were executed during the child definition. The order of execution inside the IF-THEN part is important. First the tail hidden word in the class part, probably an instance variable, is linked to the most recently defined public word in the dictionary. Second, the tail unhidden method is linked to the head method of the parent class to implement inheritance. If there is no hidden section, because Hide was not used, then the second step just overwrites the first step because #c.Pub.Tail and #c.Pvt.Tail will contain the same number. Link changes in the instance part of the new class defining object are handled similarly.

The DOES> code of a class defining object is different from that of an instance object because its data is in the name area. The handle to a class defining object's data is stored in its PFA, so the DOES> code is a fetch to get the relative address of the DFA, calculate the DFA, a dup and fetch to get the RLA of the head class method.

The Make.Instance method is simple by comparison. It creates an immediate name, stores the RLA of the head instance method in the next long word, and allots enough space for the instance data structure. The DOES> code simply duplicates the PFA and fetches the RLA of the head instance method, convert-

ing it to the LFA.

The definition of data structure and methods for instances of OBJECT are similarly defined. The only default methods provided are those useful in describing an instance object. Notice that, as with the class section, an important control variable, I.Key, is defined as a regular instance variable. This means that one could use this variable in methods, or even change its value. However, this is very dangerous since this variable is the link from between an instance and its class defining object. Changing this pointer can have disastrous results.

This essentially completes the definition of OBJECT. The actual construction is done by :OBJECT. There are a lot of references to specific names or addresses in these procedures because OBJECT is the first object.

Examples

The file concludes with several example objects. The first example is that of a simple integer object. Each instance of Integer can hold one long word integer and has the messages Save and Fetch. It is clearly very inefficient compared to the standard VARIABLE defining word. The second example shows how to define a fixed size array class of 10 cells.

The third example allows variable size array objects to be defined and shows the power of being able to redefine the Make.Instance method. This example also demonstrates use of the I.Pntr defining word. As mentioned earlier, this word must be the last instance variable and no child class may add instance variables. Thus, the instance variable Length is defined here, although it is not used by instances of Array, in order to be used by instances of a child class of Array. Be careful with this word! The next two examples use the variable array object to define a Vector object and a String object. Error checking has been implemented, but otherwise the objects are very primitive.

Finally, the example object Struct can be used to define C-style data structures. Again, it shows the power of being able to redefine the Make.Instance method. Note that even the DOES> code of Make.Instance has been changed. Also notice that the words S.Array and LongInt are instance variable defining words like Ins.Array and I.Var. These words must be defined in the instance section in order to be available to define the instance section of child class objects. These words cannot be used during definition of a child's class section, one needs to use the regular instance variable defining words.

Improvements

There are several areas in which this code can be improved. The most obvious need is for assembly coding of the object stack control words, especially the word Ocop+ since it represents the execution overhead for fetch and save operations on instance variables. It would be nice to give the object stack its own CPU register. Unfortunately MACH2 does not leave any untrashable registers for applications, unless one gives up local variables by using A2. Second, there are no dynamic objects, although the Mac's heap would make implementation relatively easy. Third, I have not included words to accommodate floating point numbers. Fourth, the error handling is rather primitive. Etc.

Caveats

I have tried to use this code as much as possible before publication. However, there certainly are combinations I have not yet tried. So, let the user beware. I would appreciate hearing about any suggestions, bugs, or improvements. My mailing address is Wayne Joerding, Department of Economics, Washington State University, Pullman, WA, 99164.

Listing 1: Object Forth for Mach2

```
\ ObjectFORTH.9  by

\    Wayne Joerding
\    S.E. 430 Dilke St.
\    Pullman, WA 99163

\ Copyright 1988 by Wayne Joerding for MacTutor

only mac
also forth

$4EBA CONSTANT JSR_d(PC)
$4E75 CONSTANT RTS

$203C        Constant  MOVE.L_#,D0
$41FB08FA    Constant  LEA_$-6(PC,D0.L),A0
$2D08        Constant  MOVE.L_A0,-(A6)

: Defer.Not.Init ." Uninitialized" abort ;
: Defer
    Create -4 ALLOT JSR_d(PC) W,
    ['] Defer.Not.Init HERE - W, RTS W,
    ;
: Let: ( - cfa+2 )  ' 2 + ;
: Do: ( cfa - )  ' over - swap W! ;

\ ==== ObjectFORTH ======================

20 4 *   CONSTANT Maxnest
VARIABLE Ostack

\ -- Allocate space for Object Stack -
Ostack 4 + maxnest + Ostack ! maxnest Vallot

GLOBAL
: Init.Ostack Ostack 4 + maxnest + Ostack ! ;

\ ========================================
CREATE ObjectFORTH

\ --- Section defining words ------
GLOBAL Defer :Class
GLOBAL Defer :Instance
GLOBAL Defer ;Class
GLOBAL Defer ;Instance
GLOBAL Defer Hide

CREATE Ob.DefWords
\ ========================================
GLOBAL
CODE DictBase@    ( - n )
    MOVE.L $-532(A5),-(A6)RTS
    END-CODE MACH

: Relink.DefWords
    [ ' Ob.DefWords body>link dup @ swap
    DictBase@ - swap ] literal literal DictBase@ + !
    ;
: Delink.DefWords
    [ ' Ob.DefWords body>link ' ObjectFORTH
    body>link over swap - swap DictBase@ - swap ]
    literal literal DictBase@ + !
    ;

\ ==== Global Variables for Class Definition ===
VARIABLE #class.Size
```

```
VARIABLE #ins.Size
VARIABLE #Forth.Head
VARIABLE #c.Head
VARIABLE #c.Pub.Tail
VARIABLE #c.Pvt.Tail
VARIABLE #i.Head
VARIABLE #i.Pub.Tail
VARIABLE #i.Pvt.Tail
VARIABLE #parent
VARIABLE #C.or.I?

\ ====== Object stack ==========================
GLOBAL
: Opush  ( n - )  \ push from Pstack to Ostack
    Ostack dup @ 4 - =
    IF ." Ostack overflow" abort THEN
    Ostack -4 over +! @ !
    ;

GLOBAL
: Opop ( - ) \ discard top number on Ostack.
    Ostack dup maxnest + 4 + swap @ =
    IF ." Ostack underflow" abort THEN
    4 Ostack +!
    ;

GLOBAL
: Ocop+ ( n - n+os )  \ add Ostack to Pstack.
    Ostack @ @ +
    ;

GLOBAL
: Ocop ( - os )  \ copy Ostack to Pstack.
    Ostack @ @
    ;


\ ==== Data Structure Defining Words =============
GLOBAL VARIABLE Offset

GLOBAL
: Ins.Array ( size - ) ( name -IN- )
\ Make a new instance variable of 'size' bytes.
    CREATE   immediate
    Offset @ ,              \ store offset
    dup 2 mod +       \ make sure even number
    Offset +!               \ increase offset
    DOES>             ( ob.DFA -OS- ob.DFA )
        @ [compile] literal
        compile Ocop+
    ;

GLOBAL
: I.Var ( - ) ( name -IN- )
    4 Ins.Array
    ;

GLOBAL
: I.Pntr ( - ) ( name -IN- )
\ Make ivar without incrementing.
    CREATE   immediate
    offset @ ,              \ compile offset.
    DOES>             ( ob.DFA -OS- ob.DFA )
        @ [compile] literal
        compile Ocop+
    ;

\ ==== Method defining word ======================
GLOBAL
: :M ( - )    \ Define a method.
    :
    ;

\ ====== Custom Abort routine ====================
80 USER (ABORT)
VARIABLE ^ABORT
: Ob.Abort ( n - )    \ Use during class definition.
    #C.or.I? @ cr
    IF ." Class part" ;Class
    ELSE ." Instance part" ;Instance
    THEN
        ^ABORT @ (ABORT) ! ABORT
    ;

\ ====== Message support ==========================
: link>name 4 + ;

: Compile.Meth    ( meth.CFA  ob.DFA - )
\
\   the following line was edited out - JL
\   [compile] literal \ make a literal of ob.DFA
\
\   and replaced by the following four lines
    MOVE.L_#,D0     W,
    here -          ,
    LEA_$-6(PC,D0.L),A0  ,
    MOVE.L_A0,-(A6)     W,

    compile Opush     \ compile a call to opush
    JSR_d(PC) W, here - W,
    compile Opop
    ;

: Do.Meth     ( meth.CFA  ob.DFA - )
    Opush execute Opop
    ;

: Do.OR.Compile   ( ob.DFA  meth.CFA  f<>0 - )
    1 =
    IF swap Do.Meth
    ELSE
        swap state @
        IF Compile.Meth
        ELSE Do.Meth THEN
    THEN
    ;

GLOBAL
: Find.Meth?
( lfa  strng.adr | f - strng.adr  CFA  f<>0 }
\ f = as with FIND
    0 -> f

    BEGIN
        lfa  link>name C@ %11111 and
        strng.adr C@ =

        IF 1 -> f strng.adr C@ L_ext 1+ 1
            DO  strng.adr I + C@ lfa 4 + I + C@ <>
                IF 0 -> f leave THEN
            LOOP
        THEN
        f 0= lfa @ 0 <> and
    WHILE    \ continue if meth <> message & LFA<>0
        lfa  @  negate +> lfa
    REPEAT
    strng.adr f
    IF lfa dup link>body swap link>name C@
        %10000000 and
        IF 1 ELSE -1 THEN
    ELSE f
    THEN
    ;

GLOBAL
: Get.Meth   ( key  strng.adr - meth.CFA f<>0 )
    Find.Meth?
    ?dup IF
```

```
        rot drop
    ELSE
        cr ." Method -> " count %11111 and
        type 3 spaces ." Not Found" ABORT
    THEN
    ;

GLOBAL
: Get.Msg    ( - strng.adr )
    32 word pad over C@ L_ext 1+ cmove pad
    ;

GLOBAL
: Selector ( ob.DFA key - )    ( <msg> -IN- )
    Get.Msg
    Get.Meth
    Do.OR.Compile
    ;


CREATE Ob.Words

\ ==== Define OBJECT ============================
: :Root
    Create            \ stopper word
    NP @ DictBase@ - ,
    0 NP @ !          \ put 0 for class.Key
    4 NP +!               \ increment NP by 4 bytes
    ;

:Root <root>

\ -- Define class section of OBJECT ------
here #c.Pvt.Tail !

0 offset !
GLOBAL   I.Var class.Key
         I.Var class.Tail
         I.Var c.Tail.link
GLOBAL   I.Var class.Size
GLOBAL   I.Var ins.Key
         I.Var ins.Tail
         I.Var i.Tail.link
GLOBAL   I.Var ins.Size
         I.Var parent
         I.Var Ob.name.link
         I.Var Class.RLA
offset @ #class.Size !

: cr5sp cr 5 spaces ;

: <Pr.Meth>   ( adr cnt - )
    cr5sp %11111 and swap over type
    25 swap - spaces ." Link adr = "
    dup . space ." Offset = " dup @ .
    ;

: Pr.meths    ( key - )
    BEGIN
        dup @
    WHILE
        dup link>name count
        <Pr.Meth> dup @  -
    REPEAT drop
    ;
CODE ClassStrucAllot  ( n - addr )
    MOVE.L   $-1FC(A5),D0
    MOVE.L   D0,D1
    ASR.L    #$1,D1
    BCC.S    @1
    ADDQ.L   #$1,D0
@1  MOVE.L   D0,A0
    ADD.L    (A6)+,D0
    MOVE.L   D0,$-1FC(A5)
    MOVE.L   A0,-(A6)
    MOVE.L   #0,-(A6)
    RTS
    END-CODE


: SetClassStruc   ( - )
    #class.Size @  ClassStrucAllot
    IF ." Memory error" . ." Handle" . abort
    ELSE
        dup DictBase@ -  ,
        Opush

        #c.Head @ DictBase@ -
                class.Key !
        #c.Pub.Tail  @ dup IF DictBase@ - THEN
                class.Tail    !

        #c.Pub.Tail @ dup
            IF dup #c.Pvt.Tail @ =
                IF #Forth.Head @ -
                ELSE @ THEN
            THEN c.Tail.link       !
        #class.Size @ class.Size   !
        #i.Head @ DictBase@ -
                ins.Key !
        #i.Pub.Tail  @ dup IF DictBase@ - THEN
                ins.Tail !
        #i.Pub.Tail @ dup
            IF dup #i.Pvt.Tail @ =
                IF #Forth.Head @ -
                ELSE @ THEN
            THEN i.Tail.link       !
        #ins.Size      @ ins.Size   !
        #parent  @ Parent           !
        last DictBase@ - Class.RLA !
        last @ ob.name.link              !
        Opop
    THEN
    ;

here #c.Pub.Tail !

:M pr.ob.ivar( - )
    cr cr ." Class instance variables are :"
    cr5sp ." class.Key      = "
        class.Key @ DictBase@ + .
    cr5sp ." class.Tail     = "
        class.Tail @ DictBase@ + .
    cr5sp ." c.Tail.link    = "
        c.Tail.link @ .
    cr5sp ." class.Size     = "
        class.Size @ .
    cr5sp ." ins.Key        = "
        ins.Key @ DictBase@ + .
    cr5sp ." ins.Tail       = "
        ins.Tail @ DictBase@ + .
    cr5sp ." i.Tail.link    = "
        i.Tail.link @ .
    cr5sp ." ins.Size       = "
        ins.Size @ .
    cr5sp ." Parent         = "
        Parent @ DictBase@ + .
    cr5sp ." ob.name.link   = "
        ob.name.link @ .
    cr5sp ." Class.RLA      = "
        Class.RLA @ DictBase@ + .
    ;
GLOBAL
:M Pr.class.meths ( - )
    cr cr ." Class methods are :"
    class.Key @ DictBase@ +
    Pr.meths
    ;
```

```
GLOBAL
:M Pr.ins.meths  ( - )
    cr cr ." Instance methods are :"
    ins.Key @ DictBase@ +
    Pr.meths
    ;

GLOBAL
:M Describe  ( - )
    cr ." — Class Information ——————————"
    cr ." NAME : " Class.RLA @ DictBase@ + dup .
    link>name count %11111 and type
    5 spaces ." pointer to class data = " 0cop .
    Pr.ob.ivar
    Pr.class.meths
    Pr.ins.meths cr
    ;

GLOBAL
:M Define.Child.Class ( - )
    last #Forth.Head !
    class.Size @ #class.Size !
    ins.Size @ #ins.Size !
    class.Key @ DictBase@ + #c.Head !
    0 #c.Pvt.Tail !
    0 #c.Pub.Tail !
    ins.Key @ DictBase@ + #i.Head !
    0 #i.Pvt.Tail !
    0 #i.Pub.Tail !
    0cop DictBase@ - #parent !
    Relink.DefWords
    [ ' Ob.Abort body>link DictBase@ - ] literal
        DictBase@ + link>body (ABORT) dup @ ^ABORT !  !
    ;

GLOBAL
:M Name.Child.Class   ( - )   ( <name> -IN- )
    Delink.DefWords
    ^ABORT @ (ABORT) !    \ restore old abort routine
    CREATE immediate
    SetCLassStruc

    \ - seal class and link to parent ———
    last dup #Forth.Head @ - swap  !
    #c.Pub.Tail @ ?dup    \ false => no class section
        IF  #c.Pvt.Tail @ dup #Forth.Head @ - swap  !
            dup class.Key @ DictBase@ + - swap  !
        THEN
    #i.Pub.Tail @ ?dup    \ false => no class section
        IF  #i.Pvt.Tail @ dup #Forth.Head @ - swap !
            dup ins.Key @ DictBase@ + - swap !
        THEN

    DOES> @ DictBase@ + dup @ DictBase@ + Selector
    ;
GLOBAL
:M Make.Instance  ( - )   ( <name> -IN- )
    CREATE immediate
    ins.Key @ ,
    ins.Size @ allot
    DOES> dup @ DictBase@ + Selector
    ;

last #c.Head !


\ —Define instance section of root object ——

here #i.Pvt.Tail !

0 Offset !

GLOBAL   I.Var I.Key

Offset @ #ins.Size !
here #i.Pub.Tail !

GLOBAL
:M Pr.Imeths ( - )
    I.Key @ DictBase@ +
    cr cr ." Instance methods are :"
    Pr.meths
    ;
GLOBAL
:M Name  ( - )
    cr ." NAME : " 0cop 4 - body>link link>name count
    %11111 and type 5 spaces ." instance.DFA = " I.Key .
    ;
GLOBAL
:M Describe ( - )
    cr ." — Instance information ——————————"
    Name
    Pr.Imeths
    ;

last #i.Head !


\ — Child Class defining words ————————
: Relink ( t o k | - ) ( dfa -OS- dfa' )
        Class.RLA @ dup DictBase@ +
        dup @ Ob.name.link  !
        swap k @ - swap  !
        o @ t @ DictBase@ + dup @ o  ! !
    ;

: Relink.Parents  ( - )
    #parent @ DictBase@ + 0push
    #C.or.I? @
    IF
        Begin
            class.Key @
        While
            class.Tail dup @
            IF      c.Tail.link  class.Key
                    Relink
            ELSE drop
            THEN Parent @ DictBase@ +
                    0pop 0push
        Repeat
    ELSE
        Begin
            class.Key @
        While
            ins.Tail dup @
            IF      i.Tail.link  ins.Key
                    Relink
            ELSE drop
            THEN Parent @ DictBase@ +
                    0pop 0push
        Repeat
    THEN
    0pop
    ;

: Delink ( o t | - ) ( dfa -OS- dfa' )
    t @
    IF
        Ob.name.link @ Class.RLA @ DictBase@ + !
        o @ t @ DictBase@ + dup @ o ! !
    THEN
        Parent @ DictBase@ +
        0pop 0push
    ;

: Delink.Parents ( - )
    #parent @ DictBase@ + 0push
    #C.or.I? @
```


© The Definitive MacTutor, Vol. 4

```
    IF
        Begin
            class.Key @  \ class.Key is zero for <root>
        While
            c.Tail.link   class.Tail
            Delink
        Repeat
    ELSE
        Begin
            class.Key @  \ class.Key is zero for <root>
        While
            i.Tail.link   ins.Tail
            Delink
        Repeat
    THEN
    Opop
    ;

: <:Class>   ( - )
    #class.Size @ Offset !
    here #c.Pvt.Tail !
    here #c.Pub.Tail !
    -1 #C.or.I? !
    Relink.Parents
    ;
Let: :Class Do: <:Class>

: <;Class>   ( - )
    last #c.Head !
    Offset @ #class.Size !
    #c.Pvt.Tail @ body>link #c.Pvt.Tail !
    #c.Pub.Tail @ body>link #c.Pub.Tail !
    Delink.Parents
    ;
Let: ;Class Do: <;Class>

: <:Instance>( - )
    #ins.Size @ Offset !
    here #i.Pvt.Tail !
    here #i.Pub.Tail !
    0 #C.or.I? !
    Relink.Parents
    ;
Let: :Instance Do: <:Instance>

: <;Instance>( - )
    last #i.Head !
    Offset @ #ins.Size !
    #i.Pvt.Tail @ body>link #i.Pvt.Tail !
    #i.Pub.Tail @ body>link #i.Pub.Tail !
    Delink.Parents
    ;
Let: ;Instance Do: <;Instance>

: <Hide>
    here #C.or.I? @
    IF  #c.Pub.Tail ! ELSE  #i.Pub.Tail ! THEN
    ;

Let: Hide Do: <Hide>

\ ── Complete and Seal root object ──────

: :OBJECT
    CREATE immediate  \ make header for "OBJECT"
    DOES> @ DictBase@ + dup @ DictBase@ + Selector
    ;

:OBJECT OBJECT

\ - Initialize temporary variables used by Set.Struc -
    ' <root> 4 + @ #parent !
    ' Ob.Words body>link #Forth.Head !
    #c.Pub.Tail @ body>link #c.Pub.Tail !
    #i.Pub.Tail @ body>link #i.Pub.Tail !
    #c.Pvt.Tail @ body>link #c.Pvt.Tail !
    #i.Pvt.Tail @ body>link #i.Pvt.Tail !

SetClassStruc          \ init class data structure

\ - seal class and link to <root> ──────
    Delink.DefWords
    ' <root> body>link #parent !
    #i.Pub.Tail @ dup #parent @ - swap !
    #c.Pub.Tail @ dup #parent @ - swap !
    #i.Pvt.Tail @ dup #Forth.Head @ - swap !
    #c.Pvt.Tail @ dup #Forth.Head @ - swap !
    0 ' <root> body>link
    last dup ' Ob.DefWords body>link - swap
    !   !

\ ====== EXAMPLES ==================
\ ====== Integer Class =============
 OBJECT Define.Child.Class
    :Instance
        I.Var Int
    Hide
        :M Fetch ( - n )
            Int @
            ;
        :M Save ( n - )
            Int !
            ;
    ;Instance

OBJECT   Name.Child.Class          Integer


\ ====== 10 cell Array Class ==============
OBJECT Define.Child.Class
:Instance
    10 Constant          Max.Size
    10 4 * Ins.Array  Head
Hide
    :M Describe
        cr ." ── Instance Information ──────────────"
        Name
        cr ." Max.Size (in cells)  = " Max.Size .
        Pr.Imeths
        ;
    :M Store ( x i - )
    \ Store value x in array for index = i
        Max.Size over 1 + < if ." index out of bounds" abort then
        4 * Head + !

        ;
    :M Retrieve ( i - )
    \ Retrieve value of array for index = i.
        Max.Size over 1 + < if ." index out of bounds" abort then
        4 * Head + @

        ;
;Instance
OBJECT   Name.Child.Class          Array10  ( - )


\ ====== Variable size Array Class ==============
Integer Define.Child.Class
:Instance
    I.Var    Max.Index     \ max size of array
    I.Var    Length        \ number of elements in array
    I.Pntr   Start         \ points to the start of array memory
Hide
    :M Describe
        cr ." ── Instance Information ──────────────"
        Name
        cr ." Max Length in cells  = " Max.Index @ 1+ .
        cr ." Cell size in bytes   = " Int @ .
```

```
        Pr.Imeths
        ;
    :M Store ( x i - ) \ Store value x in array for index = i
                       \ first cell has index of zero
        Max.Index @ over < over 0 < or
        IF ." index out of bounds" abort THEN \ <- error checking
        Int @ * Start + !

        ;
:M Retrieve ( i - )   \ Retrieve value of array for index = i.
        Max.Index @ over < over 0 < or
        IF ." index out of bounds" abort THEN \ <- error checking
        Int @  * Start + @

        ;
;Instance
:Class
    :M Make.Instance ( n c - ) ( <name> -IN- )
    \ Array instance of size n cells, cell size of c
        CREATE immediate
        ins.Key @
        ,                   \ store key to methods of parent class
        dup ,               \ save cell size in 'Int' variable
        over 1- ,               \ make and save Max.Index
        0 ,                 \ init current Length to zero
        * ins.Size @ + allot
        DOES> dup @ DictBase@ + Selector
        ;
;Class
Integer  Name.Child.Class       Array    ( n c - )

\ ====== Vector Class ================================
Array Define.Child.Class
    :Class
        :M Make.Instance ( n - )
            4 Make.Instance
            ;
    ;Class
Array Name.Child.Class          Vector   ( n - )

\ ====== String Class ================================
Array    Define.Child.Class
:Class
    :M Make.Instance ( n - )
        1 Make.Instance
        ;
;Class
:Instance
    :M Describe
        cr ." — Instance Information ——————————"
        Name
        cr 5 spaces ." Max String length        = "
                                        Max.Index @ 1+ .
        cr 5 spaces ." Current String length    = " Length @ .
        Pr.Imeths
        ;
    :M Print ( - )    \ Prints string for this instance.
        Start Length @  dup
            IF type ELSE ." string empty" drop drop THEN
        ;
    :M Store ( a - )  \ Store a string with count byte at address.
        count dup Max.Index @ 2 + <
        IF dup Length ! Start swap cmove
        ELSE cr ." String too large for 'Store' " drop drop
        THEN
        ;
;Instance
Array    Name.Child.Class       String   ( n - )

\ ==== Struc Class ==================================
OBJECT Define.Child.Class
:Class
    : Make.Instance ( - ) ( name -IN- )
        CREATE immediate
        Ins.Key @ ,
        ins.Size @ allot
        DOES> dup @ DictBase@ + Get.Msg Get.Meth
            drop execute state @
            IF [compile] literal THEN
        ;
;Class
:Instance
    : S.Array ( size - ) ( name -IN- )
        CREATE
        offset @ ,
        dup 2 mod +
        offset +!
        DOES> @  +
        ;
    : LongInt ( - ) ( name -IN- )
        4 S.Array
        ;
;Instance
OBJECT   Name.Child.Class           Struct

\ ==== Point Class ======================
Struct Define.Child.Class
    :Instance
        LongInt Xdim
        LongInt Ydim
    ;Instance
Struct   Name.Child.Class           Point
```

Listing 2: Test code for the Mach2 OOPS extensions

```
\ This was on one of Wayne's earlier disks.
\ I included it here because it still works and makes a nice example.
\ - JL
\ —— testing words ————————
\ - test Integer ——
Integer Make.Instance XY
XY Describe

\ - test of Array10 —
 Array10 Make.Instance vec10
 000 0 vec10 Store
 111 1 vec10 Store
 222 2 vec10 Store
 cr 1 vec10 Retrieve .
vec10 Describe

\ - test of Array —
 10 4 Array Make.Instance Arr
000 0 Arr Store
111 1 Arr Store
999 9 Arr Store
cr 9 Arr Retrieve .
Arr Describe

\ - test of Vector —
 10 Vector Make.Instance vec
 000 0 vec Store
 111 1 vec Store
 222 2 vec Store
 cr 2 vec Retrieve .
 vec Describe

\ - test of String —
14 String Make.Instance hello
hello Describe
: hel ." This is hello" ;
' hel 4 + hello Store
cr hello Print

\ - test of Point —
Point  Make.Instance  thisPoint
111 thisPoint Xdim ! cr thisPoint Xdim @ .
222 thisPoint Ydim ! cr thisPoint Ydim @ .
```

Forth Forum
Appletalk Mail

Jörg Langowski
MacTutor Editorial Staff
MacTutor Vol. 4 No. 9

"Appletalk mail for Mach2"

This month I'll give you a brief re-introduction to Appletalk and Mach2 words for Appletalk handling. The example will be a set of two programs - a message sender and a message receiver which together form a very rudimentary mail utility.

We've already had some contributions on Appletalk: for additional reading see the articles by Bob Denny, V1#10 and V1#11; Alan Wootton, V1#10; Dave Kosiur, V3#4; and on the printer access protocol by Mike Schuster, V2#2 and Nicholas Pavkovic, V4#1. Also, for any serious Appletalk programming it is indispensable to have a copy of Inside Appletalk handy, and the Appletalk chapter of Vols. II and V of Inside Macintosh.

I thought it worthwhile to write a sample Appletalk application in Mach2 Forth, especially since the Appletalk example included in the Mach2 system does not give a hint how to use higher level protocols.

Appletalk protocol levels

Let's - very briefly - review the way Appletalk sends data over the network.

We must view the Appletalk cable as a bus to which up to 32 devices can be attached. The bus is very simple, containing a twisted pair of wires, and information must therefore be passed along it serially. This serial transfer proceeds at a rate of 230.4K baud. The basic Appletalk message unit sent on the bus is a *packet* of data, which can be zero to 600 bytes long.

Evidently, only one device at a time can send something along the bus. If - let's say - a program instructs my Mac to send out a packet of data and at the same time the Laserwriter sitting in another room tries to send as well, everybody else will receive only garbage. The Appletalk hardware has a way of detecting this situation, and will automatically re-send the packet after a random wait. This way it is ensured that the two devices probably won't cross-talk again.

ALAP

You don't perceive this mechanism (in network language it is called the *Appletalk Link Access Protocol* or ALAP). If you ask the Appletalk driver to send out a packet of data, you can be sure that - using a transparent handshake mechanism between sender and receiver, and a CRC for error checking - it will be sent to the correct destination on the local net, or else you'll get an error message.

Devices are identified on the Appletalk line by their *node numbers*, each device having a different number. The node number is assigned automatically when a device first uses the network; it looks around for other active nodes and chooses a node number that isn't used yet. All this happens in a completely transparent way.

Packets sent out over the network are identified by their source and destination node numbers, which are contained in their header. Detection of the header is done by the hardware, so only packets addresses to one node will be noticed by that node, anything else on the bus will be ignored.

DDP

A destination node that receives a packet has to know what to do with it. Since one node might have several different usages for packets, another *protocol layer* 'on top of' the ALAP can distribute packets received in one node to different processes, called *sockets*. The protocol that ensures that a packet arrives at the right socket is called the *Datagram Delivery Protocol* (DDP).

DDP does the following: When a packet is received on a node, its data contains a *socket number* in the header. The Appletalk driver that handles DDP maintains a socket table which it searches for a match. If no match is found, the packet is simply ignored. If there is a match, control is passed to a *listener* routine which processes the rest of the data packet. This listener routine must conform to certain timing restrictions since the data keeps coming in at a rapid rate. Inside Mac Vol.II explains in detail how to write a listener routine.

DDP also handles the transfer of packets that are sent to other networks. As you probably know, several Appletalk networks may be interconnected by *bridges* to form a large *internet*. The internet might even include networks like Ethernet which use completely different data transfer mechanisms on the hardware and low-level software levels.

DDP can send a packet to an *internet address*. Such an address contains the network number, node ID and socket number of the destination. If the network number is the network that the node is on, the packet is simply sent to that node. If the node sits on another network, DDP sends the packet to a bridge on its own network (if there is one). The bridge then will know how to forward the packet. This 'routing' process and the way the bridges get to know about all the other networks on the internet is explained in great detail in Inside Appletalk. All we need to know here is that if you call DDP with an internet address that corresponds to an existing node and socket on some network, your data will (hopefully...) get there.

NBP

How does one find out whether a certain node exists on a large internet? Numbers are hard to remember, and they also can change since node IDs are always assigned dynamically, and socket numbers may be. Therefore there exists a mechanism - the

© The Definitive MacTutor, Vol. 4

name binding protocol (NBP) - to assign unique names to sockets at a node.

Any device on the network can be assigned a name. For instance, the Laserwriter called 'Laser#1' on the third floor could be called

"Laser#1:LaserWriter@3rd_floor"

where the first part of the name, up to the colon, is the actual device name, the second part, up to the '@', its type, and the final part the *zone* where the device is located.

Each network has a zone name assigned. Several networks can have the same zone name, but one network can have only one zone name. Looking for all Laserwriters in a particular zone will give the network addresses of *all* Laserwriters on *all* networks included in that zone. Bridges know about zones and the networks they include; Again you don't need to be concerned with the details *how* this is achieved. For the curious, this is well explained in Inside Appletalk's sections on the Routing Table Maintenance Protocol (RTMP) and the Zone Information Protocol (ZIP).

For the less curious it is sufficient to know that there exist Appletalk driver calls which will for instance, given a name like "Laser#1:LaserWriter@3rd_floor" return that printer's network ID, node ID and socket number, or for "=:LaserWriter@*" return a list of addresses of all Laserwriters in one's own zone ("=" for the device name or type acts as a wildcard; "*" indicates the zone of one's own network).

ATP

OK. So now we can, given the name of some device on the network, find out its address. What do we do with it?

Usually we will send some data to that device, and maybe receive some other data in return. Using DDP, this could for instance be done in the following way:

- create a socket on the node and associate to it a listener routine that will store the data received on that socket;
- send a data packet to the other device, using its network address that has been looked up using NBP. DDP will add addressing information to the data so that the receiving end knows where to send its response to.
- when a response is received from the other device, the listener routine will store the data and notify the program that data has arrived.

Note, however, that on a big internet packets might get lost, maybe some bridge breaks down or a whole network is disconnected. This might not be known to the sender at send time. Therefore, the algorithm that I just described will not necessarily ensure that the data arrives at the remote device. We would need to implement some acknowledge mechanism which re-sends a packet a certain number of times if its reception is not confirmed by the remote device after some timeout period.

Fortunately we don't have to implement such a mechanism, as it has been provided for us in the *Appletalk Transaction Protocol* (ATP) driver. The two partners in a data transmission - the requesting and the responding end - are called the *client* and the *server* and interact in the following way:

- A server opens a *responding socket* on the network using the routine *ATPOpenSocket*. ATP automatically assigns a listener routine to that socket that will dump the received data into a previously assigned buffer.
- The server will then call the routine *ATPGetRequest*. This routine will return (or, for asynchronous calls, the IOCompletion routine will be called) when a request data packet has been received.
- We assume that the client knows the network address of the server. Otherwise, the server would have to register its name on the network and the client could find the address through that name.
- The client calls *ATPSendRequest* with the network address of the server, and up to 578 bytes of request data in a buffer. This data will be transmitted to the server; ATP makes sure it gets there. *ATPSendRequest* waits for a response from the server, which can contain up to eight data packets; it will retransmit the request as long as not all data packets have been received. For the duration of the transmission and for receiving the response, *ATPSendRequest* opens a socket and automatically closes it when it returns. You, the programmer, normally don't notice this.
- The server receives the request data packet and puts it away into a buffer; *ATPGetRequest* returns and the server calls *ATPSndRsp* which transmits the response, up to eight buffers of data of a maximum of 578 bytes each. The response also contains information how many buffers in total will be sent (1 to 8), and which one is being sent in each particular packet. The client end *(ATPSendRequest)* maintains a table to know which packets have been received. All this is handled automatically by ATP.

There are even higher-level protocols (Printer Access Protocol, Appletalk Session Protocol, Appletalk Filing Protocol) which use ATP to implement particular functions. They don't concern us here, but are described in Inside Appletalk and the Appletalk chapter of IM Vol.5. We have all the necessary information to implement the example program, a very basic mail utility.

Appletalk mail example

Our example consists of two programs (stand-alone on the source code disk): the sender and the receiver. The receiver will be known on the network as <choosername>:mailbox@<zone> where <choosername> is the user name given in the Chooser dialog and <zone> the zone where the network is located.

The receiver task operates in the following way: First, it opens an ATP responding socket by calling *ATPOpenSocket*. Then, using its socket number, it makes itself known to the network with a *registerName* call to the NBP. Last, it calls *ATPGetRequest* asynchronously and puts itself to sleep (storing SLEEP in its STATUS address and calling PAUSE); the IOCompletion routine will wake it up again when the request has arrived. The request contains the mail message, which is displayed. Then, the receiver task sends a response, which consists of one buffer of data with no particular information in it. At the end, the task closes the socket and removes its name from the network. This cycle is repeated until a key is pressed.

The sender task will first search the network for any devices that match the name description "=:mailbox@*" (i.e., all open mailboxes in its own zone). It will display the list of mailboxes with their network addresses. Then it prompts for the address where the message should be sent, and for one line of message text. It calls *ATPSendRequest* with the mail message in the request data block. When it has received the response from the receiver (or has received no response, in which case it generates a timeout error), it starts the cycle again, searching for mailboxes.

I should apologize here for the very experimental implementation of this 'message service'. There are absolutely NO errors trapped, and the example will work only if you start the receiver on one Mac, then start the sender on another Mac, and then start sending mail. It will also not (yet) run in background under Multifinder; and the interface is certainly far from the User Interface Guidelines. However, the example can give you a basis on which to start your own experimentation.

Mach2 Appletalk words

After these general remarks, let's look at the Forth code. The constant definitions are taken from Inside Macintosh and from the Atalk.h header file in Bob Denny's Appletalk example from V2#10. All Appletalk routines are accessed through calls to either of two drivers: *.MPP*, which implements ALAP, DDP, RTMP and NBP, and *.ATP*, which implements ATP. Their driver IDs are 9 for .MPP and 10 for .ATP.

At the beginning of a program that uses Appletalk you should check whether port B is configured for Appletalk use; if the low four bits of system global *SPConfig* ($1FB) contain 1, this is the case. (See IM II-305). *OpenATalk* checks the ports and opens .MPP and .ATP if necessary.

All Appletalk driver calls are passed through a parameter block, ATPblock. I am always using the same global block, thus we can make at most one asynchronous call at a time. For this simple example this is sufficient. The words I wrote to call the different Appletalk functions that we need here all set up the parameter block first, then issue the driver *_Control* call through *call.atp* or *call.mpp*. I have deliberately not used assembler to set up the parameter block, but you might recode the glue in assembler to save code space.

For further details on the Appletalk calls you should refer to IM Vol II. For the *ATPGetRequest* function, we implement an asynchronous call to the .ATP driver. The word *call.atp.async* implements this function. In fact, all I/O calls that take more than a few milliseconds should be handled this way, especially in a multitasking environment like Mach2. However, for *ATPGetRequest* it is more essential than for the other functions, since that call never times out and thus would hang forever if the request is not satisfied.

For an asynchronous I/O call in Mach2, we define a 'semaphore' variable that will hold the status address of the task issuing the call. This variable is called *ATPout* (I couldn't think of a better name). *(get.request.async)* does a *ATPout get* first (ATPOut must be zero for the task to continue, then the task will put its status address into that variable), then the parameter block is set up and the asynchronous call made. After the call, the task puts the value *SLEEP* into its status address and calls *PAUSE*. This will put the task to sleep until the status value is changed back to *WAKE* again. The completion routine does exactly this. Thus execution of the task calling *(get.request.async)* will continue after the *PAUSE* when the completion routine is called. If the completion routine is never called (i.e. the task hangs), we can always stop the Mach2 system completely by typing BYE in the main window.

Some remarks on name registration: First, if you ever were curious where a program gets the Chooser user name from, it is contained in 'STR ' resource -16096 (local resource ID=0 of DRVR ID=9). *get.choosername* gets this name for you.

The calls to *register.name* and *lookup.name* are made synchronously, and they take several seconds. If that seems too long to you, you can replace those by asynchronous calls as well, but make sure to use different parameter blocks for each call to avoid big trouble. The name of a network device is contained in a special data structure called a *names table entry* or NTE, whose format is given in IM II-321 (Fig. 13). The word *make.entity* is used to construct the 'entity name' field of the NTE. When searching for network names with *lookup.name*, the list of names returned is in the format of several such entities. The words *next.field* and *print.entities* (further down in the listing) are used to traverse this list and print the names with their addresses in a readable format. Since the four-byte network address in that table may start at an odd byte, I also had to define >@<, which does a long word fetch at an odd address (I discovered this when I checked the example on a Mac+ and got an address error! There's a disadvantage using the MacII, you don't discover such things!).

Finally, while the data from an ATP request is simply put into a buffer, the data for the response must be contained in up to 8 separate buffers. What is actually passed to the *(send.response)* routine is the address of a buffer descriptor structure (BDS) containing 1 to 8 entries of 12 bytes each. The addresses of the different buffers and the length of the data are contained here (see IM II-288). The words *setup.bds* and *release.bds* are provided to get and release heap space for the ATP buffers.

Well, I'll let you look at the code for the rest of the implementation and finish with some news from my side of the Atlantic that might be interesting to Mac people on both sides.

International shareware payment - a Solidarsoft initiative -

A great problem for European users with using shareware products written in the US has always been the transfer of money across the Atlantic. Mailing checks is possible in principle, but usually the only ones that get a profit out of that are the banks with the high fees they charge for cashing foreign checks. Buying international money orders also can get expensive, and sending cash might or might not work. The second problem arises when shareware programs are used within a company or organization that needs a receipt for tax or other purposes.

It seems to be so difficult to pay shareware from overseas that the author of McSink has kindly *offered* his program to overseas users because his bank charges him more for cashing the

check than his profit would be after taking away the cost of shipping an update disk overseas!

Solidarsoft, the French non-profit organization that was founded to help improve software developer/publisher/user relationships and that I already told you about, has now come up with a scheme that greatly facilitates international shareware payment. You send a check to Solidarsoft with a letter giving the name and version of the shareware product and the author's address, or the shareware registration form, and they will handle the rest: make sure that the author gets the correct $$$ amount (by some economic way of money transfer), and that you will be correctly registered with the author. Using this service will be only slightly more expensive for a French shareware user than going to the bank, getting the paperwork done, and paying the author directly. Currently, to cover banking fees and other costs, Solidarsoft is applying the following formula:

You count the US$ at 6.50 French francs (FF). On top of that, you add 43 FF handling charge for international payments and 5 FF for payments within France. If a check does not cover the shareware fees plus handling, e.g. in the case of changing exchange rates (the dollar is going up as of this writing), it will simply be returned with an explanatory note. So to pay a $25 shareware product, you send the registration form together with a check on 205.50 FF to:

Solidarsoft,
66 boulevard Exelmans, F-75016 Paris, France.

Let's hope this example catches on elsewhere. Till next month, happy threading.

Listing 1: Appletalk handling from Mach2

```
\ Appletalk general definitions
\ 30.05.88 JL

only forth also assembler

vocabulary network
also network definitions

DECIMAL

27   constant ioPermission
18   constant ioFileName
18   constant userData
24   constant ioRefNum
26   constant csCode
28   constant socket
30   constant addrBlock

4    constant atpLoadedBit
1    constant useATalk
-97  constant portInUse
-98  constant portNotCf

     9    constant mppUnitNum
     10   constant atpUnitNum
     mppUnitNum   1+   negate
     constant mppRefNum
atpUnitNum 1+ negate
     constant atpRefNum

\ LAP defs
1    constant LAPshortDDP
2    constant LAPLongDDP
-94  constant lapProtErr
-95  constant lapExcessCollns

243  constant lapWrite
244  constant lapDetachPH
245  constant lapAttachPH

-1   constant lapOverrunErr
-2   constant lapCRCErr
-3   constant lapUnderrunErr
-4   constant lapLengthErr

\ DDP defs
5    constant ddpHdSzShort
13   constant ddpHdSzLong

1    constant ddpRTMP
2    constant ddpNBP
3    constant ddpATP

$7F  constant ddpMaxWKS
586  constant ddpMaxData
$3ff constant ddpLengthMask
128  constant ddpWKS

-91  constant ddpSktErr
-92  constant ddpLenErr
-93  constant ddpNoBridgeErr

\ CsCode values for DDP Control calls- MPP
246  constant ddpWrite
247  constant ddpCloseSkt
248  constant ddpOpenSkt

\ RTMP definitions
1    constant rtmpSkt

\ NBP definitions
$10  constant nbpBrRq
$20  constant nbpLkUp
$30  constant nbpLkUpReply
2    constant nbpSkt
15   constant nbpTupleMax
ascii = constant  nbpEquals
ascii * constant  nbpStar

0    constant ntLink
4    constant ntTuple
7    constant ntSocket
9    constant ntEntity

-1024    constant nbpBuffOvr
-1025    constant nbpNoConfirm
-1026    constant nbpConfDiff
-1027    constant nbpDuplicate
-1028    constant nbpNotFound
-1029    constant nbpNISErr

249  constant nbpLoad
250  constant nbpConfirm
251  constant nbpLookup
252  constant nbpRemove
253  constant nbpRegister
254  constant nbpKill
255  constant nbpUnload
256  constant setSelfSend

\ ATP definitions
$40  constant atpReqCode
$80  constant atpRspCode
$C0  constant atpRelCode
```

```
$20 constant atpXO
$10 constant atpEOM
$08 constant atpSTS
$02 constant atpTidValid
$01 constant atpSendChk
$3F constant atpFlagMask
$F8 constant atpControlMask

8   constant atpMaxNum
578 constant atpMaxData

249 constant atpRelRspCB
250 constant atpCloseSkt
251 constant atpAddResponse
252 constant atpSendResponse
253 constant atpGetRequest
254 constant atpOpenSkt
255 constant atpSendRequest
256 constant atpRelTCB

-1096   constant atpReqFailed
-1097   constant atpTooManyReqs
-1098   constant atpTooManySkts
-1099   constant atpBadATPSkt
-1100   constant atpBadBuffNum
-1101   constant atpNoRelErr
-1102   constant atpCBNotFound
-1103   constant atpNoSendResp
-1104   constant atpNoDataArea
-1105   constant atpReqAborted

$1FA constant pRamByte
$1FB constant SPConfig
$291 constant portBUse
$2D8 constant ABusVars
$2DC constant ABusDCE

0 constant bdsBuffSz
2 constant bdsBuffAddr
6 constant bdsDataSz
8 constant bdsUserData

.trap   _control,async   $a404

header ATPbuffer 2000 allot
header myBDS 8 12 * allot
header reqBuf 600 allot
header myNTE 100 allot
header ATPblock 50 allot

header MPPName
    DC.B 4
    DC.B '.MPP'

header ATPName
    DC.B 4
    DC.B '.ATP'

: open.atp
    PortBUse c@ $10 and
    IF 0 \ Appletalk already open!
    ELSE
        ['] ATPName ['] ATPBlock ioFileName + !
        0 ['] ATPBlock ioPermission + c!
        ['] ATPBlock call Open
    THEN
;

: open.mpp
        ['] MPPName ['] ATPBlock ioFileName + !
        0 ['] ATPBlock ioPermission + c!
        ['] ATPBlock call Open
;

: OpenATalk ( | PBUse - f )
    PortBUse c@ -> PBUse

    PBUse 0<
    IF SPConfig c@ $F AND 2 =
        IF portNotCf
        ELSE
            open.mpp ?dup 0=
            IF open.atp THEN
        THEN
    ELSE
        PBUse $F and 1 =
        IF open.atp
        ELSE portInUse
        THEN
    THEN
;

: close.atp
    ATPRefNum ['] ATPBlock ioRefNum + w!
    ['] ATPBlock call Close
;

: call.mpp
    mppRefNum    ['] ATPBlock ioRefNum + w!
    ['] ATPBlock call control
;

: call.atp
    atpRefNum    ['] ATPBlock ioRefNum + w!
    ['] ATPBlock call control
;

: call.atp.async ( p_complete - flag )
    atpRefNum        ['] ATPBlock ioRefNum + w!
    ( p_complete)['] ATPBlock ioCompletion + !
    LEA      ATPBlock,A0
    EXG      D4,A7
    _Control,Async
    EXT.L    D0
    MOVE.L   D0,-(A6)
    EXG      D4,A7
;

: open.socket( addrBlock socket# - socket# flag )
    ( socket# )  ['] ATPBlock socket + c!
    ( addrBlock )['] ATPBlock addrBlock + !
    atpOpenSkt       ['] ATPBlock csCode + w!
    call.atp
    ['] ATPBlock socket + c@
    swap
;

: close.socket ( socket# - flag )
    ( socket# )  ['] ATPBlock socket + c!
    atpCloseSkt  ['] ATPBlock csCode + w!
    call.atp
;

: (send.request)  ( userData atpFlags addrBlock
                    reqLength reqPointer
                    bdsPointer numOfBuffs
                    timeOutVal retryCount -
                  reqTID BitMap atpFlags numOfResps flag )
    ( retryCount )   ['] ATPBlock 47 + c!
    ( timeOutVal )   ['] ATPBlock 45 + c!
    ( numOfBuffs )   ['] ATPBlock 44 + c!
    ( bdsPointer )   ['] ATPBlock 40 + !
    ( reqPointer ) ['] ATPBlock 36 + !
    ( reqLength )['] ATPBlock 34 + w!
    ( addrBlock )['] ATPBlock 30 + !
    ( atpFlags ) ['] ATPBlock 29 + c!
    ( userData ) ['] ATPBlock 18 + !
    atpSendRequest   ['] ATPBlock csCode + w!
```

```
    call.atp
    ['] ATPBlock 16 + w@
    ['] ATPBlock 28 + c@
    ['] ATPBlock 29 + c@
    ['] ATPBlock 46 + c@
    4 roll ( result code )
;

: (get.request)     ( atpSocket reqLength reqPointer -
                      userData atpFlags addrBlock reqLength
                      bitMap transID flag )
    ( reqPointer )    ['] ATPBlock 36 + !
    ( reqLength )['] ATPBlock 34 + w!
    ( atpSocket )['] ATPBlock 28 + c!
    call.atp
    ['] ATPBlock 18 + @
    ['] ATPBlock 29 + c@
    ['] ATPBlock 30 + @
    ['] ATPBlock 34 + w@
    ['] ATPBlock 44 + c@
    ['] ATPBlock 46 + c@
    6 roll ( result code )
;

variable ATPout ( semaphore )

code get.request.compl
    movem.l  a0/a5,-(a7)
    movea.l  currenta5,a5
    movea.l  ATPout,a0
    move.w   #wake,(a0)
    movem.l  (a7)+,a0/a5
    rts
end-code

: (get.request.async) ( atpSocket reqLength reqPointer -
                        userData atpFlags addrBlock reqLength
                        bitMap transID flag )
    ATPout get
    ( reqPointer )    ['] ATPBlock 36 + !
    ( reqLength )['] ATPBlock 34 + w!
    ( atpSocket )['] ATPBlock 28 + c!
    ['] get.request.compl call.atp.async
    sleep status w! pause
            \ wake up when getRequest is completed
    ['] ATPBlock 18 + @
    ['] ATPBlock 29 + c@
    ['] ATPBlock 30 + @
    ['] ATPBlock 34 + w@
    ['] ATPBlock 44 + c@
    ['] ATPBlock 46 + c@
    6 roll ( result code )
    ATPout release
;

: setup.bds ( #buffers )
    0 DO
        600 call NewPtr abort" Could not get buffer memory"
        i 12 * ['] myBDS + 2+ !
    LOOP
;

: release.bds ( #buffers )
    0 DO
        i 12 * ['] myBDS + 2+ @
        call DisposPtr abort" DisposPtr failed!"
    LOOP
;

: (send.response) ( atpSocket atpFlags addrBlock
                    bdsPointer numOfBuffs bdsSize transID -
                    reqTID userData flag )
    ( transID )       ['] ATPBlock 46 + w!
    ( bdsSize )       ['] ATPBlock 45 + c!
    ( numOfBuffs )    ['] ATPBlock 44 + c!
    ( bdsPointer )    ['] ATPBlock 40 + !
    ( addrBlock )['] ATPBlock 30 + !
    ( atpFlags ) ['] ATPBlock 29 + c!
    ( atpSocket )['] ATPBlock 28 + c!
    atpSendResponse   ['] ATPBlock csCode + w!
    call.atp
    ['] ATPBlock 16 + w@
    ['] ATPBlock 18 + @
    rot ( result code )
;

: (add.response)
                    ( userData atpSocket atpFlags addrBlock
                    reqLength reqPointer rspNum transID -
                    flag )
    ( transID )       ['] ATPBlock 46 + w!
    ( rspNum )        ['] ATPBlock 44 + c!
    ( reqPointer )    ['] ATPBlock 36 + !
    ( reqLength )['] ATPBlock 34 + w!
    ( addrBlock )['] ATPBlock 30 + !
    ( atpFlags ) ['] ATPBlock 29 + c!
    ( atpSocket )['] ATPBlock 28 + c!
    ( userData ) ['] ATPBlock 18 + !
    atpAddResponse    ['] ATPBlock csCode + w!
    call.atp ( result code )
;

: load.nbp
    nbpLoad ['] ATPBlock csCode + w!
    call.mpp
;

: make.entity ( object typ zone entity | obL typL - )
    object entity     over c@ 1+ dup -> obL cmove
    typ    entity obL +   over c@ 1+ dup -> typL     cmove
    zone   entity obL + typL +  over c@ 1+           cmove
;

: (lookup.name) ( interval retry buffer size max entity -
                  matches flag )
    nbpLookup     ['] ATPBlock csCode + w!
    ( entity )    ['] ATPBlock 30 + !
    ( max )       ['] ATPBlock 40 + w!
    ( size ) ['] ATPBlock 38 + w!
    ( buffer )    ['] ATPBlock 34 + !
    ( retry )     ['] ATPBlock 29 + c!
    ( interval) ['] ATPBlock 28 + c!
    call.mpp
    ['] ATPBlock 42 + w@ \ matches found
    swap \ result code
;

: lookup.name ( object typ zone | - matches flag )
    ['] myNTE ntEntity + make.entity
    2 10 ['] ATPbuffer 600 20 ['] myNTE ntEntity +
        (lookup.name)
;

: (register.name) ( interval retry ntQEl verify -  flag )
    nbpRegister  ['] ATPBlock csCode + w!
    ( verify )   ['] ATPBlock 34 + c!
    ( ntQEl )    ['] ATPBlock 30 + !
    ( retry )    ['] ATPBlock 29 + c!
    ( interval)  ['] ATPBlock 28 + c!
    call.mpp \ result code
;

: register.name ( socket# object typ zone ) ( ntQEl | - flag )
    ntQEl ntEntity + make.Entity
    ntQEl ntSocket + c! ( store socket number )
    2 10 ntQEl 1 ( always verify ) (register.name)
;
```


© The Definitive MacTutor, Vol. 4

```
: remove.name ( ntQEl | flag )
    nbpRemove    ['] ATPBlock csCode + w!
    ( ntQEl ) ntEntity +  ['] ATPBlock 30 + !
    call.mpp \ result code
;

: set.self.send ( self_send_flag | old_flag - )
    setSelfSend ['] ATPBlock csCode + w!
    ( flag ) ['] ATPBlock 28 + c!
    call.mpp drop \ result code
    ['] ATPBlock 29 + c@
;

4ascii STR   constant "str
4ascii MAIL constant "mail

: get.choosername
    "str -16096 call getresource
    ?dup IF @ ELSE 1 abort" No Chooser name found!" THEN
;

\ ========== Forth definitions follow ============

also forth definitions

variable mailbox.socket
header mailNTE 110 allot

: open.mailbox
    0 0 open.socket abort" could not get free ATP socket"
    dup mailbox.socket !
        get.choosername " mailbox" " *" ['] mailNTE
        register.name abort" registerName failed"
;

: close.mailbox
    ['] mailNTE remove.name drop
    mailbox.socket @ close.socket drop
;

: sendOK
    cr ." — Sending OK response....."
    1 setup.bds
    " This mail was received OK." count dup ['] myBDS w!
                                  ['] myBDS 2+ @ swap cmove
    ['] myBDS    ['] ATPBlock 40 + !
    1   ['] ATPBlock 44 + c!
    1   ['] ATPBlock 45 + c!
    atpSendResponse   ['] ATPBlock csCode + w!
    call.atp
    1 release.bds
;

: get.mail ( | trID addr.block - reqTID userData flag )
    mailbox.socket @ 500 ['] reqBuf (get.request)
    abort" ATPGetRequest error!"
    5 call sysbeep
    -> trID drop ( don't need bitmap )
    cr ." ***** Mail received *****"
    cr ['] reqBuf swap type
    cr ." ***** End of mail   *****"
    dup -> addr.block
    cr ." sender: $" hex . ." , flags: $" . ." , User Data: $"
.
    cr decimal
    sendOK
;

: get.mail.async ( | trID addr.block - reqTID userData flag )
    mailbox.socket @ 500 ['] reqBuf (get.request.async)
    abort" ATPGetRequest error!"
    5 call sysbeep
    -> trID drop ( don't need bitmap )
    cr ." ***** Mail received *****"
    cr ['] reqBuf swap type
    cr ." ***** End of mail   *****"
    dup -> addr.block
    cr ." sender: $" hex . ." , flags: $" . ." , User Data: $"
.
    cr decimal
    sendOK
;

: >@< ( odd address fetch, unnecessary on MacII )
    dup 2 mod
    IF dup c@ swap 1+ @ -8 scale $FFFFFF and swap 24 scale +
    ELSE @
    THEN
;

: next.field dup c@ + 1+ ;

: print.entities ( #entities entityTable )
    cr swap hex
    0 DO ." $" dup >@< u. ."  - "
         5 + dup count type ." :"
         next.field dup count type ." @"
         next.field dup count type cr
         next.field
    LOOP drop
    decimal
;

: find.boxes
    " =" " mailbox" " *" lookup.name
        abort" LookupName failed"
    cr ?dup IF dup . ." mailbox(es) found on the network:" cr
                    ['] ATPBuffer print.entities
            ELSE ." No mailboxes found on the network." cr
            THEN
;

: send.mail ( receiver msg | - )
    cr ." Sending message to $"
        receiver hex . decimal ." ..." cr
    1 setup.bds
    "mail %00100000 receiver
    msg count swap ['] myBDS 1 2 5
    (send.request)
    ?dup IF ." SendRequest error #" . cr
        ELSE ." Mail delivered" cr
        THEN
    . . . . cr
    1 release.bds
;

\ ===== MAIL SENDER AND RECEIVER TASKS =====

also mac

NEW.WINDOW Sender
" Sender" Sender TITLE
40 20 170 400 Sender BOUNDS
Document Visible CloseBox GrowBox Sender ITEMS

400 1000 TERMINAL sendTask

NEW.WINDOW Receiver
" Mailbox" Receiver TITLE
190 20 320 400 Receiver BOUNDS
Document Visible CloseBox GrowBox Receiver ITEMS

400 1000 TERMINAL rcvTask

: mail.it
    activate
    begin
        cr ." Searching open mailboxes..." cr
```

```
        find.boxes
        hex
        begin
            cr ." To address (zero to quit): $"
                pad dup 1+ 10 expect number?
        until

        ?dup IF
            cr ." Message: "
            pad 1+ 80 expect
            span pad c!
            pad send.mail
        ELSE bye
        THEN
    again
;

: get.it
    activate
    begin
    cr ." Registering new Mailbox..."
    open.mailbox
        cr ." Waiting for mail..." cr
        get.mail.async drop
    close.mailbox
    ?terminal until
    bye
;

: setup.main
    0 ATPout !
    open.mpp drop
    open.atp drop
    1 set.self.send drop
;

: setup.sender
    setup.main
    Sender ADD
    Sender sendTask BUILD
    Sender call selectwindow
    sendTask mail.it
;

: setup.rcv
    setup.main
    Receiver ADD
    Receiver rcvTask BUILD
    Receiver call selectwindow
    rcvTask get.it
;
```

© The Definitive MacTutor, Vol. 4

Forth Forum

The Notification Manager

Jörg Langowski
MacTutor Editorial Staff
MacTutor Vol. 4 No. 10

Notification Manager, List Manager and UserItems

System updates are almost too frequent to keep track of. It has been only a couple of months since I installed release 6.0; already there are a couple of bug fixes out. I couldn't get them as of this writing, but System 6.0 has behaved stable so far, examples of ill-behaved applications are only very rare.

The scene keeps moving. As I write this, System 6.0.1 is almost released and 7.0 is being talked about; true to the old wisdom "if one version of the operating system finally behaves, go install a new one". Anyway, the example I am giving today explores a new service added in System 6.0 under MultiFinder; the notification manager.

This is one of the little things - one might say - added on the way to 'Real Inter-Process Communication', and it is responsible for the little printer icon you see flashing with the Apple symbol in the menu bar when your background printer has a problem, or the clock icon when your alarm clock goes off.

Actually, this month's program deals with many different things, the notification manager being only one of them. I wanted to create some useful utility that can keep track of appointments and remind me if something's coming up, just like the Reminder DA in one of my recent columns. However, this time I wanted to be able to enter date and time in a well-readable format, and also keep a file with the appointments that is automatically read in on startup and updated on exit. It turned out that doing these things in a nice way required some use of the list manager and a user item in a modeless dialog; so the example will show you some of those things as well.

The Plan

The program consists of a modeless dialog window which displays a list of appointments with their dates and corresponding messages. Selection, updating and scrolling of the list is handled by the list manager. The dialog allows the user to edit a list item, add and delete appointments, and exit the program.

As long as the dialog window is up, a background task (easily created under Mach2) keeps doing the following things:

- Each time the list is changed (by adding/editing/ deleting things), it gets the time and message of the next upcoming appointment from the list.
- When the system time exceeds the time of the upcoming appointment, a Notification Manager (NM) request is set up. The NM then displays an alert message with the appointment message. After the message, the program gets the next appointment from the list.
- When the main dialog window is closed, the background task writes the appointment list back to the file and terminates the program.

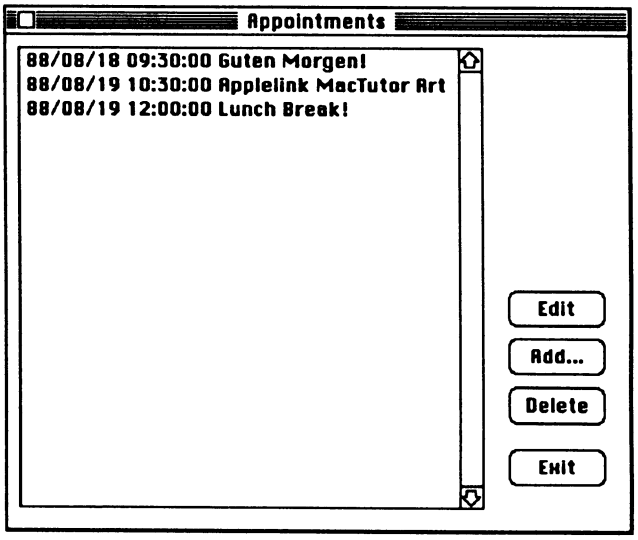


Figure 1. Main dialog window of this month's example

Implementation

First of all, we need to create a modeless dialog window that contains a list. Since that is not a standard dialog item, it has to be implemented as a user item. Just as a reminder: a user item is anything in a dialog that has to be drawn using our own procedure when the dialog is updated. A pointer to the drawing procedure is passed to the dialog manager by calling SetDItem, as described in TN34:

First call:

```
GetDItem (DialogPtr, itemNo, type, item, box);
```

check if necessary whether type is really the type of a userItem, and then call

```
SetDItem (DialogPtr, itemNo, type, @drawProc, box);
```

where @drawProc is a pointer to the user-supplied draw procedure. Note that it is really a pointer that is passed here, not a handle like in the case of other dialog items. DialogPtr, of course, is the pointer to the dialog window, where the item itemNo has been defined as a userItem, and box is the rectangle where the item will be drawn.

We can install a user item procedure from Mach2, but, if we just passed any Mach2 routine's pointer to the SetDItem trap, the machine would crash gloriously when trying to draw the item, because the procedure is not called from the Mach2 context, but through the dialog manager. This means, glue code has to be written–something you should be used to by now.

In the example, the routine gUser provides this service, setting up space on the stack for the local Forth and DO...LOOP stacks, saving all the registers, and moving the dialog pointer and item numbers on the Forth stack. It then calls the UserDraw procedure and cleans up after it has returned.

List Manager

Our user item will contain a list. How do we handle it? IM Vol.IV describes the List Manager routines, and using those the job becomes pretty easy. I am not going to review the List Manager in detail; IM gives a pretty good description, and we also had an article last year describing the list manager (V3#4).

For the drawing procedure, all we need is a call to *LUpdate*, which redraws the list. This is implemented in UserDraw (see Listing). However, here we assume the list has already been setup. This is done at the start of the program, calling installUserDraw which calls MakeList. The latter routine (taken from Palo Alto Shipping's ListManager example) creates a list in a given rectangle and window with vertical and horizontal cell size given in pixels. Looking at the example you will also notice that our list has a vertical scroll bar; the list manager handles scrolling automatically. Note that the list rectangle was inset by one pixel with respect to the user item rectangle, except for the right side where the scroll bar appears, where it had to be inset by 16 pixels (the scroll bar is drawn outside the list). After the list is created, its handle is stored in a variable for further use.

We now need a handler for mouse down events in the user item; and also routines that will allow us to edit, add and delete list items when the appropriate buttons in the modeless dialog are pushed. The words userList-handler, userEdit, userAdd, and userDelete are provided for this purpose.

Mouse down events in the user item are handled by userList-handler. It calls *LClick* with the local mouse position, the modifiers word of the event record and the list handle. (Note that event-record is a Mach2 system variable which points to the latest event record). *LClick* handles selection of list elements, highlighting, scrolling, and double clicks. If a list item is double-clicked, the userEdit procedure is called; otherwise, the routine just returns, with the list selection changed.

userEdit and userAdd set up a modal dialog which allow a date and text to be entered into a message field. While userAdd adds a new field at the end of the list, userEdit takes the current selection, puts the date and message into the modal dialog, and lets the user edit it. getMsg is the routine that parses the message string into its date and message fields and displays the edit dialog. update-cell writes the current date/message string into the selected cell. userDelete simply deletes the currently selected cell.

A number of things could be improved here: mainly, I did not implement automatic sorting by date when a list item is added or edited. This is straightforward and would require one or two more pages of code; since it is not really essential for the operation, I'm leaving it as an exercise for you.

The list is initially filled from a file called '*dates*' which has to be in the same folder as the application. If it doesn't exists, it will be created. *dates* contains the date/message strings, separated by carriage returns, and can be edited like any text file. fill-list opens - or creates - the file, reads the strings and adds rows to the list. At the end of the program, write-list writes the list back to the file. This way, by making the program one of your Multifinder startup applications, you will be automatically reminded of any upcoming appointment that was entered into the list.

Modeless Dialog Handling In Mach2

What has to be done to handle modeless dialogs correctly in a multitasking environment like Mach2? Since events are handled for us by the I-O task, we need not - and should not - call *WaitNextEvent*, *IsDialogEvent*, or *DialogSelect* from our program. This has already been done for us. All we need to do to connect a modeless dialog window with one particular task is install the task pointer in the dialog window's *refCon* field. The I-O task will look at the front window's *refCon* and pass the event information to the owning task. In the case of a dialog event it will put the dialog pointer and the item number into the user variables DialogHandle and DialogData (offsets 140 and 136). When data is stored there, the owning task will run the modeless dialog handling routine whose pointer is stored in the user variable modelessVector (offset 132). This routine gets passed the item number and the dialog pointer. Our program uses the word dialog-handler to call the handlers for the individual dialog items. The pointer to this routine is installed on starting up the task; likewise we install a pointer to a menu handling routine, which allows to handle desk accessories, a simple file menu which can only quit the program, and an Edit menu which allows for editing in desk accessories (in case you ever want to run this program from the finder, which is not very likely). Setting up the menus and menu handlers should be self-explanatory from the listing.

The Main Program

The main program window is hidden behind the menu bar, its coordinates being (1,1,16,16). This is because Mach2 can only assign a menu bar to a terminal task, therefore we need a main window. I have not tried whether - since we also have the dialog window associated with the task - the main window can be simply made invisible by calling *HideWindow*, so I simply used the 'tiny-window-behind-menu-bar' method that one would use in Mach2 for having menus without a window.

The main task loop does two things:

- checks whether an appointment has 'matured' and must be displayed using the Notification Manager (check_next_date);
- checks whether the dialog window is still visible. If it is not, the user has closed it and the program should write the list back to the file and exit (check_dialog_up).

A flag, nmChanged, is kept to indicate whether check_next_date should search the list for the next appointment due or simply compare the time of the last appointment read with the system time. The flag is set by the list editing routines, and by check_next_date itself when it has displayed an appointment. It is reset by the routine that gets the next appointment from the list. check_next_date calls say_it when the system time has exceeded

the time of the appointment. say_it (finally!) calls the Notification Manager.

Notifying the User

The actual notification routine, as it turns out, is the smallest part of the program and of my column. say_it first checks the flag, nmPresent, which has been set at startup time to indicate whether the NM traps are present. If not, we're running under an older system version, and say_it simply beeps. Otherwise, it calls notify-request, our glue routine to the *NMInstall* trap. The NM traps are explained in TN184, and I'll repeat some of the information here.

notify-request takes the following parameters from the stack:

mark, contains 1 if the application should be marked in the Apple menu, 0 if not, or the reference number of a desk accessory to mark that desk accessory.

SIcon, contains a handle (non-purgeable) to a SICN resource if a small icon should flash with the Apple icon in the menu bar, 0 otherwise.

sound, contains a handle to a sound record to be played with *SndPlay* if a special sound is to be played on the notification; 0 if no sound should be made and -1 for the system beep.

str, a pointer to a string to appear in the NM alert box, or 0 if no alert.

resp, a pointer to a response procedure. This is explained in more detail in TN184; we simply use -1 to indicate the standard system response procedure which simply removes the NM request from the queue. If 0 is passed, no response procedure is called, and the program itself would be responsible for removing the NM request using *NMRemove*.

refCon, a constant available for general use, TN184 suggest to store A5 of the calling program here to allow access to the application globals.

myNMRec will contain the NM record in the format given in TN184, and *NMInstall* is called passing its address in A0.

That's it; the alert will come up when an appointment is due if you let the 'appts' application (provided on the source code disk) turn in the background. Try to experiment with the other NM options, the small icons and sounds.

One last remark: in some cases it might happen that not the actual appointment is displayed in the alert box, but the *next one* on the list. This happens when it takes the NM longer than one second to get the resources and display the alert. In that case, the background task will already have continued and put the next message string in the msgTxt global variable! This can be remedied by increasing the wait period (presently, 60 ticks) after calling say_it in check_next_date.

The necessary resources which have to be added to the MACH.RSRC file for running the program are given in Listing 2, in Rez format. Don't forget to add a SIZE -1 resource as well, with the *canBackground* bit set, and a size of approx. 120K. Good luck.

Mach 2.14 (beta)

The latest news is a beta version 2.14 of Mach2, which I received recently; as you read this, 2.14 might be released, and I'll give you a list of the main changes in one of the next columns. Here just a very brief summary:

- compiler optimization has been improved.
- local variable handling has been greatly improved, with the possibility to reserve blocks of local variable space (i.e. for parameter blocks), and customizing the local variable compiler.
- some words have been added, among them SHIFT which takes the function of SCALE that I previously defined.
- more trap words have been added to the system. The NM traps, which I defined here, are included.
- the disassembler has been enhanced, with listing of references to USER and global variables.

That's it; till next month.

Listing 1: Notification Manager example

```
\ notification manager example System Ver 6.0 needed
\ JL 15.8.88

only forth definitions
also assembler also mac also i/o
decimal

\ structure of a NM record

0 CONSTANT qLink     \ pointer
4 CONSTANT qType     \ integer
6 CONSTANT nmFlags   \ integer
8 CONSTANT nmPrivate  \ longint
12CONSTANT nmReserved \ integer
14CONSTANT nmMark \ integer
16CONSTANT nmSIcon  \ handle
20CONSTANT nmSound  \ handle
24CONSTANT nmStr    \ StringPtr
28CONSTANT nmResp \ ProcPtr
32CONSTANT nmRefCon \ longint

8 CONSTANT nmType

.TRAP  _NMInstall $A05E
.TRAP  _NMRemove  $A05F

CODE NMInstall  ( NMRec - result )
  MOVE.L (A6)+,A0
  _NMInstall
  MOVE.L D0,-(A6)
  RTS
END-CODE MACH

CODE NMRemove ( NMRec - result )
  MOVE.L (A6)+,A0
  _NMRemove
  MOVE.L D0,-(A6)
  RTS
END-CODE MACH

$5E CONSTANT nmTrap#
$9F CONSTANT unkTrap#

variable myNMRec 32 vallot
variable nmPresent
      \ for checking whether the NM is implemented
variable nmChanged
      \ flag for telling supervisor task
      \ that something has changed
variable nmSecs
      \ time in seconds for next notify alert
```

```
: notify-request
  ( mark SIcon sound str resp refCon | -
          nmPtr result )

    nmType mynmRec qType + w!
    mark   mynmRec nmMark + w!
    SIcon    mynmRec nmSIcon + !
    sound    mynmRec nmSound + !
    str      mynmRec nmStr + !
    resp   mynmRec nmResp + !
    refCon mynmRec nmRefCon + !

    mynmRec dup NMInstall
;

300 CONSTANT AppleID
301 CONSTANT FileID
302 CONSTANT EditID

2000 CONSTANT updID
2001 CONSTANT msgID

110 CONSTANT wVisible \ offset into window record

132 USER modelessVector
 60 USER fID

CREATE APPLESTRING  $01 C,  $14 C,

NEW.WINDOW nmWindow

" NM" nmWindow TITLE
1 1 16 16 nmWindow BOUNDS
Plain Visible NoCloseBox NoGrowBox nmWindow ITEMS

600 4000 TERMINAL nmTask

NEW.MBAR nmBar

NEW.MENU AppleMenu
APPLESTRING AppleMenu TITLE
0 APPLEID AppleMenu BOUNDS
" About Appointments ...;(-" AppleMenu ITEMS

NEW.MENU FileMenu
" File" FileMenu TITLE
0 FileID FileMenu BOUNDS
" Close;Quit"FileMenu ITEMS

NEW.MENU EditMenu
" Edit" EditMenu TITLE
0 EditID EditMenu BOUNDS
" (Undo/Z;(-;Cut/K;Copy/C;Paste/V;Clear" EditMenu ITEMS

VARIABLE DAName 60 VALLOT
VARIABLE updStore 160 VALLOT
      \ for 'update appointments' dialog
VARIABLE updPtr
VARIABLE msgPtr \ for storing the dialog pointers
VARIABLE updRect 4 vallot
VARIABLE hUpdList \ stores list handle
VARIABLE listRows   \ total # of rows in list
VARIABLE datim 10 VALLOT
        \ 14 bytes for date-time record
VARIABLE msgTxt 252 VALLOT
        \ 256 bytes for item text

\ ***** list manager support

\ List Manager select flags.
128 CONSTANT OnlyOne
 64 CONSTANT ExtendDrag
 32 CONSTANT NoDisjoint
 16 CONSTANT NoExtend
  8 CONSTANT NoRect
  4 CONSTANT UseSense
  2 CONSTANT NoNilHilite

\ Offsets into the List record
 12 CONSTANT IndentOffset
 36 CONSTANT SelFlags
 80 CONSTANT LDataHandle

CREATE ArrayDim \ Initially we will have an empty array.
        \ We'll add rows and columns later.
  0 W,   \ Row-o.
  0 W,   \ Column-o.
  0 W,   \ Row-i.
  1 W,   \ Column-i.

: NewVList ( rview databounds size wPtr - lhandle )
  0    \ LDEF proc id
  swap \ window pointer
  0    \ DrawIt flag
  0    \ HasGrow flag
  0    \ scrollHoriz flag
  -1   \ scrollVert flag
  (CALL) LNew
;

: MakeList ( rect vcell hcell wPtr |
      lhandle cellpt rectbr recttl - lhandle )
  rect @ $10001 + -> recttl
  rect 4 + @ $10010 - -> rectbr
  ^ recttl      \ Pass rectangle.
  ArrayDim      \ Pass bounds.
  vCell ^ cellpt W!
  hCell ^ cellpt 2+ W!
  cellpt        \ Pass cell size.
  wPtr
  NewVList -> lhandle

  OnlyOne NoNilHilite +
          \ Select only one cell at a time.
  lhandle @ SelFlags + C!
          \ Don't hilite empty cells.
  lhandle
;

: readline ( ^pfile string | pStr char - flag )
  string -> pStr
  BEGIN
    ^pfile @ virtual c@ -> char
    1 ^pfile +!
    char 0= char 13 = OR 0= WHILE
    1 +> pStr
    char pStr c!
  REPEAT
  pStr string - string c!
  char
;

: open-dates-file
  " Dates" $open dup 0<
  IF drop " Dates" dup
      $create drop
      $open
  THEN
  fID w!
;

: fill-list ( | pfile theCell - )
  0 listRows !
  open-dates-file
  0 -> pfile
  BEGIN
    ^ pfile msgTxt readline WHILE
```


© The Definitive MacTutor, Vol. 4

```
    1 listRows @ hupdList @ call LAddRow drop
    Ø -> theCell
    listRows @ ^ theCell w!
    1 listRows +!
    msgtxt count theCell hupdList @ call LSetCell
  REPEAT
  fID w@ closefile
  -1 hUpdList @ call LDoDraw
  hupdList @ call LAutoScroll
;

: write-list ( | len theCell offset - )
  open-dates-file
  Ø -> offset
  listRows @ Ø DO
    Ø -> theCell i ^ theCell w!
    Ø -> len  255 ^ len w!
    msgTxt ^ len theCell hUpdList @
      call LGetCell
    ^ len w@ -> len
    13 msgTxt len + c!
    1 +> len
    offset len msgTxt fID w@ write
    len +> offset
  LOOP
  Ø msgtxt c!
  offset 1 msgTxt fID w@ write
  offset 1+ fID w@ setEOF
  fID w@ closefile
;

\ UserDraw procedure
\ must use (call) instead of call and use glue code
\ for saving registers and setting up Forth stack

: UserDraw ( theDlg theItem | iType iHdl rectbr recttl - )

  theDlg theItem ^ iType ^ iHdl ^ recttl
      (call) GetDItem
  ^ recttl (call) FrameRect
  theDlg 24 + @ \ visRgn of dialog window
  hUpdList @    \ list handle
    (call) LUpdate
;

\ UserDraw procedure glue code
\ sets up local stack etc.

CODE gUser
  LINK A6,#-512   ( 512 bytes of local Forth stack )
  MOVEM.L A0-A5/D0-D7,-(A7)    ( save registers )
  MOVE.L A6,A3    ( setup local loop return stack )
  SUBA.L #256,A3  ( in the low 256 local stack bytes )
  CLR.L  D1
  MOVE.W 8(A6),D1   ( theItem )
  MOVE.L 10(A6),D0    ( theDialog )
  MOVE.L D0,-(A6)
  MOVE.L D1,-(A6)

  UserDraw

  MOVEM.L (A7)+,A0-A5/D0-D7  ( restore registers )
  UNLK A6
  MOVE.L (A7)+,A0    ( return address )
  ADD.W  #6,A7       ( pop off 6 bytes of parameters )
  JMP    (A0)
  RTS
END-CODE MACH

: update-cell ( string | theCell - )
  Ø -> theCell
  -1 ^ theCell hupdList @ call LGetSelect
    IF
      string count theCell hupdList @
        call LSetCell
    THEN
;

: getText ( dlgPtr item# string | iType iHdl iBox - string )
    dlgPtr item# ^ iType ^ iHdl ^ iBox
      call GetDItem
    iHdl string call GetIText
    string
;

: setText ( dlgPtr item# string | iType iHdl iBox - )
    dlgPtr item# ^ iType ^ iHdl ^ iBox
      call GetDItem
    iHdl string call SetIText
;

: setup-msg ( editDlg string | - string )
      editDlg  8 string getText ( year )
      editDlg  9 string 3 + getText ( month )
      editDlg 10 string 6 + getText ( day )
      editDlg 11 string 9 + getText ( hour )
      editDlg 12 string 12 + getText ( min )
      editDlg 13 string 15 + getText ( sec )

      editDlg 3 string 18 + getText ( message )
              c@ 18 + string c!

      ascii / dup string 3 + c! string 6 + c!
      32 string 9 + c! 32 string 18 + c!
      ascii : dup string 12 + c! string 15 + c!
      string
;

: parse-msg ( string | sPtr - )
  string c@ 18 - string 18 + c!
  6 Ø do
    string i 3 * + -> sPtr
    2 sPtr c!
    sPtr call stringtonum
    datim i 2* + w!
  loop
  datim w@ 1900 + datim w!
;

: set-dlg ( editDlg string | - )
  editDlg  8 string setText ( year )
  editDlg  9 string 3 + setText ( month )
  editDlg 10 string 6 + setText ( day )
  editDlg 11 string 9 + setText ( hour )
  editDlg 12 string 12 + setText ( min )
  editDlg 13 string 15 + setText ( sec )
  editDlg 3 string 18 + setText ( message )
;

: getMsg ( text | editDlg itemHit iTyp iHdl iBox
      - string_or_zero )
  msgID Ø -1 call GetNewDialog -> editDlg
  text parse-msg
  editDlg text set-dlg
  Ø ^ itemHit call ModalDialog
  ^ itemHit w@ CASE
    1 OF editDlg msgTxt setup-msgENDOF
    2 OF ( Cancel ) Ø ENDOF
  ENDCASE
  editDlg call DisposDialog
;

: userEdit ( | theCell len - )
  Ø -> theCell
  -1 ^ theCell hupdList @ call LGetSelect
  IF
    255 ^ len w!
    msgTxt 1+ ^ len theCell hupdList @ call LGetCell
```

```
    ^ len w@ msgtxt c!
    msgtxt getMsg ?dup IF
      update-cell
    THEN
  THEN
  nmChanged on
;

: userAdd  ( | theCell - )
  " yy/mm/dd hh:mm:ss Your message - "
  dup c@ 1+ msgtxt swap cmove
  msgtxt getMsg IF
    0 -> theCell
    -1 ^ theCell hupdList @ call LGetSelect
    IF 0 theCell hUpdList @ call LSetSelect THEN

    1 listRows @ hupdList @ call LAddRow
    0 -> theCell
    listRows @ ^ theCell w!
    1 listRows +!
    msgtxt count theCell hupdList @ call LSetCell
    -1 theCell hUpdList @ call LSetSelect
    hupdList @ call LAutoScroll
  THEN
  nmChanged on
;

: userDelete ( | theCell - )
  0 -> theCell
  -1 ^ theCell hupdList @ call LGetSelect
  IF
    1 ^ theCell w@ hupdList @ call LDelRow
    -1 listRows +!
  THEN
  nmChanged on
;

: userList-handler ( | thePt thePort - )
  ^ thePort call getPort
  call frontwindow call setport
  ^ thePt call getMouse
  thePt event-record modifiers + w@
    hUpdList @
    call LClick
  IF ( double click ) userEdit THEN
  thePort call setPort
;

: CloseMe
  updPtr @ call CloseDialog
;

: QuitMe CloseMe ;

: dialog-handler
  ( itemHit dlgPtr | - )

  itemHit CASE
    1 OF CloseMe ENDOF
    2 OF userEditENDOF
    3 OF userAdd ENDOF
    4 OF userDelete ENDOF

    5 OF userList-handler ENDOF
  ENDCASE
;

: installupdPtr
  updPtr @ ?dup IF
    nmTask @ 2+ call SetWRefCon
  ELSE
    cr ." Couldn't create dialog"
    ABORT
  THEN
;

: installuserDraw ( pUser | iType iHdl rectbr recttl - )
  updPtr @ 5 ^ iType ^ iHdl ^ recttl
      call GetDItem
  updPtr @ 5 ^ iType w@ pUser ^ recttl
      call SetDItem
  ^ recttl 16 280 updPtr @ MakeList
        hUpdList !
;

: UndoMe ;
: CutMe ;
: CopyMe ;
: PasteMe ;
: ClearMe ;

: do-about
  128 0 CALL Alert DROP
;

: do-apple   ( item# )
  \ item# = 1 (About...)?
  item# 1 =
  IF do-about
  ELSE
    Applemenu @ item# DAName CALL GetItem
    DAName CALL OpenDeskAcc DROP
  THEN ;

: DO-FILE ( item# -  )
  ( handles selections from the file menu )
  CASE
  1 OF CloseMe    ENDOF
  2 OF    QuitMe    ENDOF
  ENDCASE
;

: DO-EDIT ( item# - )
  ( handles selections from the edit menu )
  dup 1- call SysEdit ( item# flag )
  0= IF
    CASE
    1 OF UndoMe      ENDOF
    3 OF CutMe       ENDOF
    4 OF CopyMe      ENDOF
    5 OF PasteMe       ENDOF
    6 OF ClearMe       ENDOF
    ENDCASE
  THEN
;

: nmBAR-handler  ( item# menuID -  )
  CASE
  APPLEID OF DO-APPLE ENDOF
  FILEID  OF DO-FILE  ENDOF
  EDITID OF DO-EDIT ENDOF
  ENDCASE
  0 CALL HILITEMENU
;

\ setup notify request with text in (msgTxt + 18)

: say_it
  nmPresent @ IF
    1 ( mark )
    0 ( no Icon )
    -1 ( system beep )
    msgTxt 18 + ( string to display )
    -1 ( remove request )
    0 ( no refCon )
    notify-request
    2drop
  ELSE
```


© The Definitive MacTutor, Vol. 4

```
    5 call sysbeep
  THEN
;

: get_time ( | time - secs )
  ^ time call readdatetime drop @
;

: get_next_date ( | len theCell secs - )
  listRows @ 0 DO
    0 -> theCell i ^ theCell w!
    255 ^ len w!
    msgTxt 1+ ^ len theCell hupdList @ call LGetCell
    ^ len w@ msgtxt c!
    msgtxt parse-msg
    datim call date2secs -> secs
    get_time secs u< IF leave THEN
    -1 -> secs \ in case no date matches
  LOOP
  secs nmSecs !
  nmChanged off
;

: wait ( ticks | time - )
  call tickcount ticks + -> time
  begin
    pause
    call tickcount time >
  until
;

: check_next_date
  nmChanged @ IF get_next_date THEN
  nmSecs @ get_time
  u< IF
    say_it
    60 wait
    nmChanged on
  THEN
;

: check_dialog_up
  updPtr @ wVisible + c@
  0= IF write-list bye
  THEN
;

: go.nm
  ACTIVATE
  fill-list
  ['] dialog-handler modelessVector !
  ['] nmBar-handler menu-vector !
  nmWindow dup call showWindow
    call selectWindow
  updPtr @ dup call showWindow
    call selectWindow
    call DrawMenuBar
  begin
    PAUSE
    check_next_date
    check_dialog_up
  again
;

: nmPresent?
    NMTrap# CALL GetTrapAddress
    UnkTrap# CALL GetTrapAddress
    = IF 0 ELSE 1 THEN
    nmPresent !
;

: start
  nmPresent?
  nmWindow ADD
  nmWindow nmTask BUILD

  nmBar ADD
  nmBar AppleMenu ADD
  AppleMenu @ ascii DRVR CALL AddResMenu
  nmBar FileMenu ADD
  nmBar EditMenu ADD
  nmBar nmTask mbar>task

  updID updStore -1 call GetNewDialog
  updPtr !

  installupdPtr
  ['] gUser installUserDraw

  nmChanged on
  nmTask go.nm
;
```

Listing 2: Resources for example (MPW Rez format)

```
resource 'DITL' (2000, "date list") {
  { /* array DITLarray: 5 elements */
    /* [1] */
    {272, 328, 296, 392},
    Button {
      enabled,
      "Exit"
    },
    /* [2] */
    {168, 328, 192, 392},
    Button {
      enabled,
      "Edit"
    },
    /* [3] */
    {200, 328, 224, 392},
    Button {
      enabled,
      "Add..."
    },
    /* [4] */
    {232, 328, 256, 392},
    Button {
      enabled,
      "Delete"
    },
    /* [5] */
    {8, 8, 312, 312},
    UserItem {
      enabled
    }
  }
};

resource 'DITL' (2001, "appt text") {
  { /* array DITLarray: 15 elements */
    /* [1] */
    {160, 256, 184, 320},
    Button {
      enabled,
      "OK"
    },
    /* [2] */
    {160, 336, 184, 408},
    Button {
      enabled,
      "Cancel"
    },
    /* [3] */
    {49, 10, 145, 410},
    EditText {
      disabled,
      ""
```

```
    },
    /* [4] */
    {15, 275, 33, 295},
    StaticText {
      disabled,
      ":"
    },
    /* [5] */
    {15, 235, 33, 255},
    StaticText {
      disabled,
      ":"
    },
    /* [6] */
    {15, 129, 33, 149},
    StaticText {
      disabled,
      "/"
    },
    /* [7] */
    {15, 89, 33, 109},
    StaticText {
      disabled,
      "/"
    },
    /* [8] */
    {16, 58, 32, 86},
    EditText {
      disabled,
      ""
    },
    /* [9] */
    {16, 101, 32, 127},
    EditText {
      disabled,
      ""
    },
    /* [10] */
    {16, 140, 32, 167},
    EditText {
      disabled,
      ""
    },
    /* [11] */
    {16, 205, 32, 231},
    EditText {
      disabled,
      ""
    },
    /* [12] */
    {16, 244, 32, 271},
    EditText {
      disabled,
      ""
    },
    /* [13] */
    {16, 284, 32, 312},
    EditText {
      disabled,
      ""
    },
    /* [14] */
    {16, 8, 32, 50},
    StaticText {
      disabled,
      "Date:"
    },
    /* [15] */
    {16, 176, 32, 196},
    StaticText {
      disabled,
      "at"
    }
  }
};

resource 'DITL' (128, "About Appts") {
  { /* array DITLarray: 2 elements */
    /* [1] */
    {159, 82, 188, 151},
    Button {
      enabled,
      "Okay <Ø>"
    },
    /* [2] */
    {35, 28, 128, 192},
    StaticText {
      disabled,
      "Appointment reminder - \nNotification Man"
      "ager example\n© 1988 J. Langowski / MacTu"
      "tor"
    }
  }
};

resource 'DLOG' (2000, "date list") {
  {54, 38, 380, 446},
  noGrowDocProc,
  invisible,
  goAway,
  Øx0,
  2000,
  "Appointments"
};

resource 'DLOG' (2001, "appt text") {
  {48, 16, 248, 440},
  dBoxProc,
  visible,
  noGoAway,
  Øx0,
  2001,
  "Edit Appointment"
};

resource 'ALRT' (128, "About Appts") {
  {40, 40, 240, 280},
  128,
  { /* array: 4 elements */
    /* [1] */
    Cancel, visible, silent,
    /* [2] */
    Cancel, visible, silent,
    /* [3] */
    Cancel, visible, silent,
    /* [4] */
    Cancel, visible, silent
  }
};
```

Forth Forum
Code Segments & Linker

Jörg Langowski
MacTutor Editorial Board
MacTutor Vol. 4 No. 11

Code segments and a Mach 2 linker

This month I'd like to report on some recent Mach2 improvements: the new 2.14 version and a linker for kernel-independent applications that has recently appeared on the GEnie Mach2 roundtable. Since many of you don't have access to that BBS, I'll document the linker here, with some review of code segment structure, and put the code on the source code disk.

Single-segment linker

We have seen several examples - DAs, XCMDs, MDEFs - of Mach2 code that runs independent of the Forth multitasking kernel. Writing such code requires that the programmer write the setup code that is usually provided by the Mach2 system. For the examples that I gave in my column, the standard glue code for making a routine callable from outside Mach2 looked similar to:

```
CODE prelude
    LINK A6,#-Nstack
        \ Nstack bytes of local Forth stack
    MOVEM.L A0-A5/D0-D7,-(A7)      \ save registers
    MOVE.L A6,A3 \ setup local loop return stack
    SUBA.L #Nlocal,A3 \ in the low Nlocal stack bytes
    MOVE.L 8(A6),D0   \ pointer to parameter block
    MOVE.L D0,-(A6)
    RTS \ just to indicate the MACHro stops here
END-CODE MACH

CODE epilogue
    MOVEM.L (A7)+,A0-A5/D0-D7      \ restore registers
    UNLK A6
    MOVE.L   (A7)+,A0        \ return address
    ADD.W    #4,A7    \ pop off 4 bytes of parameters
    JMP      (A0)
    RTS
END-CODE MACH

: my-forth-code
    ( code to be called externally )
;

: ext.routine
    prelude my-forth-code epilogue
;
```

After these definitions, the routine *ext.routine* may be called from the outside as if defined in Pascal as:

procedure ext.routine (parameter:**longint**);
begin
(code to be called externally)
end;

All we need in this case before calling our Forth code is to set up a local Forth stack maintained by the A6 register, save all the registers for safety, and create a loop return stack maintained by A3.

In principle, a complete application can be created this way; however, some more setup is required. A simple one-segment application consists of two CODE resources: the jump table in CODE 0 and the actual code in CODE 1. The structure of the jump table (JT) as given in IM II-60 looks like the following (in the case that the first entry in the JT corresponds to a routine in segment 1):

```
0:  longint      Above A5 size ( 32 + length of JT )
4:  longint      Below A5 size (appl. and QD globals)
8:  longint      Length of jump table in bytes
12: longint      Offset from A5 to jump table (32 )
16: Jump table:
    -- Jump table entry #1 --
    word         offset of routine #1 from beginning of
                 segment
    longint      MOVE.W #1,-(A7)
                 ( push segment # of routine on stack)
    word         _LoadSeg
    --   following jump table entries --
    --   for routine #2...n, if necessary --
```

When an application is launched, the CODE 0 resource will be loaded and the first JT entry executed. This will load the appropriate segment into memory and jump to the routine to which the first entry is pointing. Thus, a simple one-segment application would consist of a CODE 0 resource like above, with one single JT entry:

```
$nnnn          ( entry address in CODE 1 segment )
      ( attention: segment starts at )
                 ( beginning of resource + 4)
$3F3C0001      ( MOVE.W     #1,-(A7) )
$A9F0          ( _LoadSeg )
```

CODE 1 would contain the actual code, written in Mach2.

What are the advantages of creating single-segment Mach2 programs? First of all, we can create very small applications. The smallest conceivable application, which does absolutely nothing but return, would comprise only 30 bytes:

```
CODE 0:
      $00000028      ( always $20 + length(JT) )
      $00000200      ( arbitrary )
      $00000008      ( length of JT; one entry )
      $00000020      ( always )
      $0000          ( entry address in CODE 1 segment )
      $3F3C0001      ( MOVE.W     #1,-(A7) )
      $A9F0          ( _LoadSeg )
CODE 1:
      $0000          ( first routine is at beginning of JT )
      $0001          ( one entry in this segment )
      $4E75          ( RTS )
```

This program is enclosed on the source code disk as a curiosity. The file actually is 364 bytes long (Resource map etc.), which is still pretty small.

A second advantage is that we have complete control over the way the application sets itself up. In particular, we could pass a routine pointer to *_InitDialogs* to activate the Resume button of the system bomb box, or we might want to control the amount of calls to *_MoreMasters*.

The disadvantage of compiling applications under Mach2: Obviously we have to care about all the things that the kernel

normally does for us, like basic event handling, menu and menu bar setup, screen input/output, etc. Particularly, there are quite a few Forth words that may not be used anymore; regular readers of this column should be familiar with the rules for creating 'kernel-independent code' that I've laid out a few times already.

The Forth words that *can* be used include:

! " + +! ^ +> - -> 0< 0= 0> 1+ 1- 2* 2+ 2- 2/ 2DROP 2DUP 2OVER < <> = > >BODY >R ?DUP @ ABS AND ASCII C! C@ DROP DUP EXIT I I' J LEAVE L_EXT NEGATE NOT OR OVER PAD PICK R> R@ SWAP U< W! W@ XOR { (it's OK to use local variables)

The following control and branching structures may also be used:

IF ELSE THEN BEGIN WHILE REPEAT UNTIL AGAIN CASE ENDCASE OF ENDOF DO LOOP +LOOP

Assembler, of course, may be used freely.

Waymen Askey, of Palo Alto Shipping, has created a 'linker' utility that compiles single-segment application using the strategy given above. We reprint his program in listing 1 with his permission. This linker operates on a program which has the following structure:

```
PROGRAM programname;
( definitions not to be included in the final application
such as constants, compiling words, etc. )

VAR
( global variable declarations will be offset from A5 )

PROCEDURES
( Forth words called by the top level word )

MAIN
( top level word which is called on startup. )
( This word should call the setup procedures )
( MachSetUp and MacintoshSetUp )
( which are provided with the Linker utility. )
END
```

The linker computes the 'below A5' size from the variables defined after the VAR statement, adding space for the Forth stacks, Quickdraw globals and various other things. The offset of the MAIN entry point into the code segment is calculated and the jump table set up. *MakeJumpTable* and *MakeMain* are the words that create the jump table and code segment 1.

MachSetUp initializes the registers for Mach2 usage. Floating point (D7), parameter (A6) and return (A3) stacks are created above the current stack base in the application globals area. The A7 stack, starting at *CurStackBase*, remains unaffected. The application globals area is then cleared.

MacintoshSetUp does the standard initialization calls to *_MoreMasters*, *_InitGraf*, *_InitFonts*, *_InitWindows*, *_InitMenus*, *_TEInit*, *_InitDialogs*, *_FlushEvents* and *_InitCursor*.

After these initialization calls, the main program may be entered. An example of a short program which creates a window and beeps is given in the listing. This program, too, is only 858 bytes long (!!!).

Mach 2.14 upgrade

For those of you who haven't yet upgraded to Mach2.14, I'll briefly review the latest changes.

1. CASE optimization: redundant instruction sequences of the type

```
MOVE.L      D0,-(A6)
MOVE.L (A6)+,D0
```

are no longer generated.

2. Local variable handling: the new release offers access to the local variable compiler with the words LALLOT and LP@. For example, a word might define a local 16-byte buffer in the following way:

```
: EXAMPLE  {  |  [ 12 LALLOT ] myBuffer -  }
   CR  ." Please enter your name "
   ^ myBuffer  16 EXPECT
   CR ." Hello "  ^ myBuffer  SPAN @  TYPE ;
```

The local variable compiler can be further enhanced through 'local variable compiling words'; examples on how to do this are given on the 2.14 release disk.

3. Disassembler: References to USER and global variables are now given with their Forth names. Disassembly speed has been greatly improved, which is particularly evident when executing IL on a Mac Plus or SE. 68881 opcodes are now supported, however, 68020-specific instructions not yet.

4. New words: *ASCII* now takes up to four characters, for easy definition of resource types. *4+*, *4-*, *4**, *4/* have been added. *a n SHIFT* will shift a 32-bit word *a* by *n* bits.

5. The trap list has been updated.

Feedback dept.

"Dear Jörg,

I saw a discussion of accented character problems in the July MacTutor and thought I would throw in a few digressions on that matter.

First, you and your readers might be interested to know that Apple has removed the scaron and zcaron characters from the new NTX PROMs against the recommendation of Adobe. These characters were not accessible from the keyboard, because they are uncoded, meaning that no ASCII value is assigned. The only way to access them is via Postscript character names. The good news is that several new characters were added making the NTX almost compliant with the ISO 8859 character set that Adobe routinely supplies with all new fonts. (Apple removed the ´y and ´Y, too).

For those of you who want to see the unencoded characters, you can get at them with the following Postscript code and a download utility, if you have one of the new unprotected Adobe fonts or a late model Laserwriter Plus with v.3 PROMs:

```
/Garamond-Light findfont dup length dict
    /newdict exch def
```

```
(1 index /FID
    ne( newdict 3 1 roll put )( pop pop )ifelse
    } forall
/Encoding 256 array def
Encoding 0 /Garamond-Light findfont
/Encoding get 0 256 getinterval putinterval
Encoding 127 /DEL put
Encoding 129 /lslash put
Encoding 130 /Lslash put
Encoding 131 /eth put
Encoding 132 /Eth put
Encoding 133 /thorn put
Encoding 134 /Thorn put
Encoding 135 /onehalf put
Encoding 136 /onequarter put
Encoding 137 /threequarters put
Encoding  138  /brokenbar  put
Encoding 139 /onesuperior put
Encoding 140 /twosuperior put
Encoding 141 /threesuperior put
Encoding 142 /scaron put
Encoding 143 /Scaron put
Encoding 144 /zcaron put
Encoding 145 /Zcaron put
Encoding 146 /yacute put
Encoding 147 /Yacute put
newdict /Encoding Encoding put
/IsoGaramond newdict definefont pop
/IsoGaramond findfont 18 scalefont setfont
75 250 moveto
(                    Garamond) show
showpage
```

Unfortunately there is no way, at present, to get at these with templates in Fontographer, so you have to make your own composites, if you want to add these characters to PostScript fonts.

Best regards, Tim Ryan
SourceNet
P.O.Box 6767
Santa Barbara, CA 93160

PS: The standalone caron (ˇ) is frequently found as ASCII character 255, one of the last four untypeable characters. It can be accessed using QUED, ... and MS Word if you enter the character using its ASCII value.

By the way, I did get a Greek + Hebrew System from Apple-France via persistent phone calls.

I've enclosed the first draft of an article that will appear in my forthcoming book "The Macintosh Book of Fonts". If you're interested in reprinting the final draft when it's available, let me know.

Tim "

Thanks, Tim, for that interesting letter (I've enclosed your Postscript code on the source code disk). Now if all these characters were defined somewhere in the standard fonts, wouldn't that be nice? I always wondered why there were so many empty places in the font definition tables, seems like a waste of space to me...

In the next issue we'll introduce - with other contributions from this side of the Atlantic - a very nice and powerful utility for changing keyboard definitions, so at least that problem can be overcome. Till then.

Listing 1: Mach 2.14 single-segment linker

```
\ © Waymen Askey c/o Palo Alto Shipping
\ Reprinted with permission. - JL

\ Guidelines for use of the single-segment "linker."
\ This utility is NOT meant to replace the
\ standard Mach TURNKEY process.  Its use (at present)
\ is limited to creating small (one-segment, less than
\ 32K) programs which do NOT require the multi-tasking,
\ I/O, and  auto event-handling support which the normal
\ turnkey process supplies.
\ Also, since this utility is being supplied free to
\ Mach users, Palo Alto Shipping will NOT assume
\ responsibility for support of the utility, nor
\ will we be held responsible for any errors (bugs)
\ which it may produce.
\ It should, however, point the way for other
\ compiler enhancements by users.  The "bottom line" is
\ that Mach can be used to create any type (and size) of
\ Macintosh application, DA, driver, INIT, etc.
\ Waymen @ PASC
(
\ The following words MAY be used freely within the
\ stand-alone, "linked" application.
\
!  "  +  +!  -  +)  -  -)  0(  0=  0)  1+  1-  2*  2+  2-
2/  2DROP  2DUP  2OVER  (  ()  =  )  )BODY  )R  ?DUP  @
ABS  AND  ASCII  C!  C@  DROP  DUP  EXIT  I  I'  J  LEAVE
L_EXT  NEGATE  NOT  OR  OVER  PAD  PICK  R)  R@  SWAP
U(  W!  W@  XOR  {  (it's OK to use local variables)
    \ The following control and branching structures MAY
\ also be used.
\
IF  ELSE  THEN  BEGIN  WHILE  REPEAT  UNTIL  AGAIN
CASE  ENDCASE  OF  ENDOF  DO  LOOP  +LOOP

\ All assembler words may be used.
\  The following compilation words MAY be used to
\ create your application  (but don't attempt to
\ compile them, they can't be executed during the
\ run-time of your finished application).
:  ;  VARIABLE  CONSTANT  USER  CREATE  DOES)
;CODE  CODE  END-CODE  ALLOT  VALLOT  ,  W,  C,
HERE  COMPILE  [COMPILE]  IMMEDIATE  SMUDGE  LITERAL
LAST  MACH  RECURSIVE  [  ]
\ Note, global variables may ONLY be used if you
\ declare a VAR block.

\ ['] should be used with caution.
\ Don't use it on words defined outside of your
\ program block.
\ If you wish to use EXECUTE, it may be redefined as
\
CODE EXECUTE  ( a - )
  MOVE.L (A6)+,A0
  JSR (A0)
  RTS
END-CODE MACH
\ ONLY the following MAC vocabulary words MAY be used.
\ Remember to use  (CALL)  instead of  CALL.
\
(CALL)  All CONSTANTS used for the creation of user
interface structures (CLOSEBOX, VISIBLE, etc.)
If you define a VAR block, EVENT-RECORD (and all
other system global variables) may be
used as a storage area only - events (and other
information) will NOT automatically be posted
there).
\ These utilities may also be used.
TRAP#  TRAPLIST  TRAPNAME


\ ==================================
```

```
\ ========== Can't Use These ============
\ Words which may NOT be used!!!  This is NOT a
\ complete list, just some of the more common words.
\ You must NOT compile any word which is referenced
\ through Mach's own jump table (words which compile
\ a JSR d(A5) instruction.
\
CALL  GLOBAL  TERMINAL  TURNKEY  NEW.WINDOW  ADD (etc.)
BUILD  TASK  TASK->  BYE  EVENT-TABLE  PAUSE
All I/O such as  KEY  EXPECT  EMIT  ."  TYPE  (etc.)
<#  #  #S  #>  DEPTH  2SWAP  CMOVE  *  /  /MOD  */MOD  */
(if you are using a Mac II exclusively, you may substitute the
new 32-bit math routines which came with the last Mach
upgrades.)  NO SANE words,  NO TALKing words,
NO I/O words.  None of the "high-level" FILE words in
the MAC vocabulary.  NO words which reference the
multi-tasking kernel, NO I/O task words (i.e. events
MUST be handled explicitly, you must create your own
event-loop).
A space for USER variables is reserved for your
program, but all of them (except for the TIB value,
S0, and RETURN_STK) are initialized to zero.
Consider USER variables as just another global storage
area.  Words like BASE and (ABORT) may be used; however, they
will have NO effect on your program unless you specifically
design the words to use them.
)

\ ------------------------------------
\ A simple, one-segment linker which may
\ be used (with restrictions) to create
\ small applications in high-level Forth.
\ Also allows you to create very small assembly
\ language programs (mininum size about 40 bytes)
\ With slight modifications to the "linker" and the
\ proper SetUp word, could also be used
\ to create DA's, FKEY's, XCMD's, and INIT's.
\ - Waymen
\ @ Palo Alto Shipping Company

ONLY MAC ALSO FORTH DEFINITIONS
DECIMAL

$908     CONSTANT CurStackBase
$434F4445     CONSTANT 'CODE'
$4150504C     CONSTANT 'APPL'
$3F3F3F3F     CONSTANT '????'
%1          CONSTANT MainErr
%10      CONSTANT EndErr
%100     CONSTANT ProcErr
$12344320     CONSTANT GoodStart
$1234432F     CONSTANT StartFlag
$12344328     CONSTANT GoodEnd

\ The default stack and USER variable sizes
\ to be used in building the jump table.
\ I've made the USER size larger to allow
\ for a 256 byte PAD
572      CONSTANT USERSize ( USER variables)
74       CONSTANT TIBSize       ( plus STATUS)
600      CONSTANT ParameterSize( A6 & A3 stacks)
200      CONSTANT FPSize        ( FP stack)
206      CONSTANT GrafSize      ( QD globals)

$20      CONSTANT BL
-1       CONSTANT TRUE
0        CONSTANT FALSE

VARIABLE VarEntry
VARIABLE SegmentEntry
VARIABLE MainEntry
VARIABLE SegmentEnd
VARIABLE ProgramFlag
VARIABLE ProgramName  28 VALLOT
VARIABLE JumpTable    20 VALLOT


: -Leading  ( addr cnt | whiteSpace - addr' cnt' )
\ Adjusts addr and cnt to "strip" leading spaces from a
string.
\ Addr is the starting character address,
\ cnt is the original length.
    0  -> whiteSpace
    BEGIN
        addr whiteSpace +  C@  BL =
        whiteSpace cnt <  AND
    WHILE
        1  +> whiteSpace
    REPEAT
    addr whiteSpace +
    cnt whiteSpace - ;

: RemoveSpaces  ( addr | cnt  - )
\ Given counted string at addr, remove trailing and leading
\ spaces and repack string.
    addr COUNT  -TRAILING  addr C!  DROP
    addr COUNT  -Leading  -> cnt
    ( addr') addr 1+  cnt  CMOVE  cnt addr C! ;

: Scan  ( addr num delimiter | cnt char - flag )
\ Scans input stream, placing characters into string at addr
until
\ num characters are received or delimiter is found.
\ If delimiter is NOT found prior to num, return FALSE
\ else return TRUE.
    num 0>
    IF
        0  -> cnt
        BEGIN
            0 WORD  1+  C@  -> char
            char  delimiter = NOT
            num cnt  >  AND
        WHILE
            1  +> cnt  char  addr cnt +  C!
        REPEAT
        cnt addr C!  char delimiter =
    ELSE
        0 addr !  FALSE
    THEN ;

: PROGRAM  ( | cnt scanFlag - )
\ Gets program name and init's linker variables.
    ProgramName  31  ASCII ;  Scan  -> ScanFlag
    ProgramName RemoveSpaces

    ProgramName C@ 0=  scanFlag 0=  OR
    ABORT" Must use  ; to delimit program name!"

    0 MainEntry !  0 SegmentEnd !  0 VarEntry !
    StartFlag ProgramFlag ! ;

: ClearErr  ( errNum - )
    ProgramFlag @  XOR  ProgramFlag  ! ;

: VAR  ( - )
\ Ensure that current VP offset from A5 is
\ even, then save it.
    VP @   1 AND
    IF
        1 VALLOT
    THEN  VP @  ABS VarEntry ! ;

: Globals?  ( - )
\ Checks to see if a VAR statement was made.
    VarEntry @  0=
    IF
        10 CALL SysBeep
        CR ." WARNING: No global variables were declared!"
```


© The Definitive MacTutor, Vol. 4

```
    THEN ;

: ?HERE  ( - a )
\ Ensures that HERE pointer is even, then
\ returns HERE.
    HERE  1 AND
    IF
        1 ALLOT
    THEN  HERE ;

: PROCEDURES  ( - )
    ProcErr  ClearErr
    ?HERE SegmentEntry !  4 ALLOT ;

: MAIN  ( - )
    MainErr ClearErr
    ?HERE  MainEntry ! ;

: END  ( - )
    EndErr  ClearErr
    ?HERE  SegmentEnd ! ;

: ZeroFlags  ( - )
    0 ProgramFlag !  0 VarEntry !
    0 MainEntry !  0 SegmentEnd ! ;

: BelowA5  ( - n )
\ Calculates the Below A5 space for
\ the jump table.
    VarEntry @  DUP  0=
    IF
        DROP  GrafSize
    THEN
    USERSize +  TIBSize +
    ParameterSize +  FPSize + ;

: MakeJumpTable  ( - handle f )
\  handle is to a generic, one-entry jump table.
    $00000028     JumpTable         !    \ Above A5 size
    BelowA5       JumpTable 4 +!
                            \ Global variable space
    $00000008     JumpTable 8 +!
                            \ Jump table length
    $00000020     JumpTable 12 +    !
                            \ Jump table A5 offset

    \ Calculate segment entry point
    MainEntry @  SegmentEntry @  4 +  -
    ( entry) JumpTable 16 +    W!
    $3F3C0001     JumpTable 18 +    !
                            \ MOVE.W #1,-(A7)
    $0001A9F0     JumpTable 22 +    W!
                            \ _LoadSeg
    JumpTable  24  CALL PtrToHand ;

: MakeMain  ( - handle f )
    \ Offset to first jump-table entry
    0  SegmentEntry @  W!
    \ Only one jump-table entry
    1  SegmentEntry @  2+  W!
    SegmentEntry @                          ( start of
segment )
    SegmentEnd @  SegmentEntry @  -     ( length of segment )
    CALL PtrToHand ;

: Link  ( refNum | JumpHandle  MainHandle - )
\ Creates, then adds CODE segments 0 and 1
\ to file refNum
    refNum
    IF
        MakeJumpTable
        IF
            ZeroFlags
            refNum  CALL CloseResFile
            CR ." MakeJumpTable error!"  ABORT
        THEN
        -> JumpHandle
        JumpHandle  'CODE'  0  " Jump Table"
        CALL AddResource
        CALL ResError
        IF
            ZeroFlags
            refNum  CALL CloseResFile
            JumpHandle  CALL DisposHandle  DROP
            CR  ." Link (0): AddResource error!"  ABORT
        THEN

        MakeMain
        IF
            ZeroFlags
            refNum  CALL CloseResFile
            JumpHandle  CALL DisposHandle  DROP
            CR ." MakeMain error!"  ABORT
        THEN
        -> MainHandle
        MainHandle  'CODE'  1  " Main"
        CALL AddResource
        CALL ResError
        IF
            ZeroFlags
            refNum  CALL CloseResFile
            JumpHandle  CALL DisposHandle  DROP
            MainHandle  CALL DisposHandle  DROP
            CR ." Link (1): AddResource error!" ABORT
        THEN
    THEN ;

: CreateApplFile  (  | refNum -  refNum or zero )
\ Remember to delete previously made files!
    0  -> refNum
    '????' 'APPL' ProgramName  0  CreateFile
    DISK 4 +  W@  0=
    IF
        ProgramName  CALL CreateResFile
        ProgramName  CALL OpenResFile
        \ This logic returns either a valid refNum or zero,
        \ as OpenResFile returns a -1 if it can't open the
file.
        DUP  -1 =  NOT AND  -> refNum
    THEN  refNum ;

: ?Error  (  errFlag -  )
\ Checks for proper program headings
    errFlag  GoodEnd  XOR
    IF
        CR  ." Missing: "
        errFlag $FFFFFFF0 AND  GoodStart  =
        IF
            errFlag %111 AND
            CASE
                MainErr  OF   ." MAIN "      ENDOF
                EndErr  OF    ." END "  ENDOF
                ProcErr  OF   ." PROCEDURES "
                              ENDOF
                ( else)
                ." MAIN, END and/or PROCEDURES "
            ENDCASE
        ELSE
            ." PROGRAM "
        THEN
        ." Statement(s)!"  ZeroFlags  ABORT
    THEN ;

: MakeApplication  (   | refNum - )
    ProgramFlag @  ?Error
    CreateApplFile  -> refNum
    refNum
    IF
```

```
        refNum Link  Globals?
        refNum  CALL CLoseResFile
        ZeroFlags
    ELSE
        CR ." CreateFile error #"  DISK 4 + W@  L_EXT  .
        ZeroFlags  ABORT
    THEN ;

\ ==========================================
\ All of the code previous to here will NOT be included
\ in the linked program.  Thus, the above utilities may be
\ workspaced, used and/or enhanced at will.
\ MachSetUp and MacintoshSetUp should appear as the
\ first statements in your MAIN or LAUNCH word.

CODE MachSetUp  ( - )
\ Sets up stacks for high-level Forth.
\ Not needed if you work only in assembly language.
    MOVE.L CurStackBase,D0
    MOVE.L D0,D1
    ADD.L #FPSize,D0
    MOVE.L D0,D7  \ FP stack
    MOVEA.L D0,A3 \ "loop" stack
    ADD.L #ParameterSize,D0
    MOVEA.L D0,A6 \ parameter stack
    ADD.L #TIBSize,D0
    MOVEA.L D0,A4 \ USER variables
    MOVEA.L D1,A0
    MOVE.L A5,D0
    SUB.L D1,D0       \ below A5 bytes to clear
    DIVU.W #16,D0
    MOVE.W D0,D2 \ "blocks" to clear
    SWAP.W D0         \ bytes to clear
    \ Init all globals, USER
    \ vars and stack area to zeros
    BRA.S @20
@10 CLR.L (A0)+
    CLR.L (A0)+
    CLR.L (A0)+
    CLR.L (A0)+
@20 DBF D2,@10
    BRA.S @40
@30 CLR.B (A0)+
@40 DBF D0,@30
    \ Although it can't really be used,
    \ here I set-up the (TIB) USER var
    MOVE.L A6,24(A4)
    MOVE.L A6,4(A4)       \ S0 USER var
    MOVE.L A3,12(A4)  \ RETURN_STK USER var
    RTS
END-CODE MACH

CODE MacintoshSetUp  ( - )
    _MoreMasters
    _MoreMasters
    PEA -4(A5)
    _InitGraf
    _InitFonts
    _InitWindows
    _InitMenus
    _TEInit
    CLR.L -(A7)
    _InitDialogs
    MOVE.L #$0000FFFF,D0
    _FlushEvents
    _InitCursor
    RTS
END-CODE MACH

\ ==========================================
\ ============= An Example ====================

\ From this point on (between the PROCEDURES and END
\ statement) is where you place your application code.

PROGRAM My Example;
\ The required and beginning statement in your program.
\ The application will be titled as whatever appears
\ between the PROGRAM statement and the delimiting
\ colon (up to 31 characters).
    \ Words defined here
    \ will NOT be included in your application
    \ The redefintion of CALL is just a reminder.
: CALL
    CR ." Don't use CALL here, use (CALL) instead."
    ABORT ;

    0 CONSTANT NIL
    -1 CONSTANT InFront
    10 CONSTANT TenTicks
    30 CONSTANT HalfSecond
VAR
\ All global variables used within the program MUST follow
\ the VAR statement.  If you don't include the VAR
\ statement, a warning will be given during program link.
\ If you don't use global variables (or Mach system
\ globals), you may ignore the warning.
    VARIABLE DelayTicks
    VARIABLE BoundsRect 4 VALLOT
PROCEDURES
\ All subroutines must appear between PROCEDURES and
\ MAIN. Only that code appearing between the
\ PROCEDURES and END statements will appear in your
\ finished application.

    : SetDelay ( n - ) DelayTicks ! ;
    : Beeper  ( beepTime - )
        HalfSecond  TenTicks
        DO
            beepTime  (CALL) SysBeep
            I SetDelay
            DelayTicks @  (CALL) Delay  DROP
        TenTicks +LOOP ;
    : MakeWindow  ( - a | returns a window pointer)
        BoundsRect 20 72 492 322  (CALL) SetRect
        NIL  BoundsRect  " Beeper Window" VISIBLE
        NOGROW  InFront  NOCLOSEBOX  NIL
        (CALL) NewWindow ;

    : ProgramLoop  ( | windowPointer - )
        NIL  -> windowPointer
        MakeWindow  -> windowPointer

        10 Beeper

        windowPointer 0= NOT
        IF
            10  (CALL) SysBeep
            windowPointer " BYE"  (CALL) SetWTitle
            60  (CALL) Delay  DROP
            windowPointer  (CALL) DisposWindow
        THEN ;
MAIN
\ The program's entry point must appear immediately
\ after  MAIN
    : LAUNCH  ( - )
        \ Don't attempt to use local variables in the
        \ LAUNCH word. The stacks aren't created until
        \ after MachSetUp.
        MachSetUp
        MacintoshSetUp
        ProgramLoop ;
END ( of program "My Example")
\ This statement does error checking
\ and creates the application.
MakeApplication

CR .( An application called "My Example" has been created.)
```

Förth Forum

European Developers' Conference

Jörg Langowski
MacTutor Editorial Staff
MacTutor Vol. 4 No. 12

Letter from Europe

Happy Holidays. As MacTutor is entering its fifth year, we see more and more activity on the "other" (my) side of the atlantic, and many great articles have been sent to my address which we just couldn't publish yet. Coincidentally, the first European developers' conference has just been held in Paris; so we decided to dedicate part of this issue to the European Mac developers' scene.

First I would like to express my thanks to all of you who have contributed to this issue and waited patiently for your article to appear. We are looking forward to more great contributions from overseas - keep them coming in!

European Developers' conference, Paris

If Apple's figures are correct (9000 certified developers in the US and 1000 in all of Europe), quite a significant proportion of those 1000 showed up at the conference. The main lecture hall was packed with 600 people, and there was just not enough time to listen to all the talks or see all the exhibits. This report is therefore necessarily selective, but I hope I can give you an overview anyway.

This being the first developers' conference I attended and the first such event in Europe, I cannot draw any comparisons. To me it seemed rather focused on marketing rather than technical aspects of development. [Apple's developer support right now seems to emphasize commercialization of "almost-ready" projects; more and more talented Mac programmers that I know start having difficulties getting certified at an early stage]. *[Guess what Jörg; US Developer's feel the same emphasis by Apple US on marketing and completion of "almost-ready" projects. -Ed]*

The list of speakers included mainly people from Apple Europe, but also a few famous names from the outside scene. All in all, it was a well-organized event, at a conference center in a park at the city limits of Paris, with good food included; one could meet quite a few people [for instance, long-standing MacTutor contributor Steve "Prof. Mac" Brecher and I met for the first time in person] and learned a couple of new things.

The main drawback of the conference was its lack of time. 2 days packed with lectures and exhibits on the developers' fair changing twice a day (there were only 20 stands for 80 exhibits) was just too much, and Apple took a little too much time for its slick visionary presentations. Which were, anyway, interesting to listen to. Some highlights: "UNIX is an operating system neither my nor your wife would use - or want to" (Mike Spindler). "There is no artificial intelligence"; "A real personal computer is a computer you'd want to take to bed"; "When we'll finally succeed in replacing the mouse with a pen, those used to three-button mice will have to take saxophone lessons" (Jean-Louis Gassée). However, to make room for those presentations which I'd file under "entertainment", the technical talks had to be distributed into two parallel sessions, and I couldn't split myself, so I had to miss quite a bit.

Neil Minkley, of the European Developer Services Group, emphasized that international compatibility still left a lot to be desired; for instance, most character recognition programs won't deal with accented characters - a simple neglect on the part of developers who are not used to thinking "multinational". Language, he said, might be considered an attribute of a piece of text, just as script, font or style. That way one could greatly simplify hyphenation, spelling checkers, etc. The perspective of a European market without trade barriers after 1992, should encourage this kind of development. Neil summarized his talk in one more visionary quote, saying we should not have to wait till then - why not 1989.

Interesting to note was that French developers will finally get access to AppleLink. Not so hot: the access level will probably only allow for sending and receiving memos, all the other services won't be accessible for a while. Whatever sense that makes.

We also had a chance to get some "first hand" information on the Mac IIx by Peter Orr, who made (involuntarily) a convincing point against upgrading for Mac II owners. It's not quite clear to me how one can manage to squeeze a 30-minute talk out of a one-page press release, but he did. At least I learned nothing new compared to what I had read already, and simply a 68030 and 68882 at 16 MHz didn't seem that revolutionary to me. An interesting question - which was not addressed - was if and when the Multifinder will finally *require* the PMMU. From all the promises about compatibility that were made at the conference, I can only hope-guess that we won't have to trash our Pluses and SEs for a while. *[The Mac IIx in my mind is strickly a marketing gimick to take the heat off the introdudction of the Next machine, which is vastly a better price/performance point than the Mac IIx, at Apple's inflated prices. My guess is that the real Mac IIx, with the proper high end clock speed, will be introduced next year and that machine will have amuch better price/performance point in comparison to Next. So what is the point of buying the IIx now, if Apple already plans to up the clock speed in six months? -Ed]*

Very thorough presentations were given on the two mainframe connection products, MacAPPC and MacWorkstation. Fergus Murphy, who showed MacAPPC, outlined the IBM LU 6.2 networking protocols and then described the intelligent APPC communication card developed by Apple in great depth. This card sits in a Mac II, contains its own 68000 and operating system and will provide the APPC network service to any Macintosh on the same Appletalk network.

Alain Andrieux presented MacWorkStation, the program that will provide a front end to any application running on a mainframe connected to the Mac by a network or serial link. With my programs still running on a VAX and having a full Mac "look and feel", this seems to be the ideal toy for me. Later, on the Apple Expo, I could watch two demos: one was an application especially written for MacWorkStation, with menus, dialog boxes and window appearance controlled by the mainframe application, and one "plain vanilla" interface to a database where only few changes could be made to the program and therefore only text was displayed in various windows. Setup of the various Mac interface elements was amazingly fast, even though the mainframe connection was only at 1200 baud.

Alain Andrieux also gave a talk about MPW 3.0, which should be - hopefully - available in France by the time you read this. The points that impressed me most at first glance were the new symbolic debugger, SADE, which allows source-level debugging with Smalltalk browser-like windows across different programming languages under MPW, and the perspective that tools (like compilers) will finally run in the background under Multifinder. As a whole, it was confirmed that the Mac system is moving towards 32-bit addressing, and distinction between user and supervisor modes. We're looking toward some more compatibility headaches here...

Speaking of compatibility, Jim Friedlander from Apple Cupertino gave a long list of do-s and don't-s. I guess since there still exist some programmers who write directly into the menu bar, such talks are justified. A couple of things in the discussion were interesting to note: The old Graf3D library won't be supported anymore (who would have guessed?), the SANE engine won't be speeded up much, they still think (or at least they say) precision is more important than speed, and Macintalk won't exist for languages other than English unless one of the foreign developers writes another version.

With all the do-s and don't-s, I missed the why-s. Developer guidelines seem to have a way to be stated rather than justified. Now, since modern software technology is largely object-oriented, and object orientation has to do a lot with information hiding, I kind of understand Apple's way of thinking - to hide some of the information behind the developer guidelines that is considered "too dangerous" for the lowly "restuvus". Rest assured that MacTutor will continue to try its best to break this information barrier. *[The same thing happened in the Apple II world. After the days of Steve Wozniack hauling piles of loose-leaf documentation to user group meetings, Apple went through a period of denying the technical details of the machine, asserting that the unwashed masses didn't need to know all those gory details, and that revealing them would somehow tarnish the image of the appliance computer. When the Woz came back into the company, and the Apple IIgs was introduced, a lot of that nonsense went by the wayside. -Ed]*

Developers' fair

Parallel to the conference, an exhibition was held to give participants a chance to show their products. Well, at least for three hours. Since space was so limited, each exhibit was shown only half a day, then someone else took over the booth. I was able to see at least some things.

The Macintosh with its easy localization is especially adapted to a multi-lingual environment like Europe, and there were numerous examples of products making use of these properties.

Winsoft - our Grenoble local heroes - showed their multi-lingual multi-script word processing system, Wintext. With a word-processing functionality somewhere between MacWrite and MS Word, this program will allow you to mix left-to-right and right-to-left writing, e.g. in combined English/Arab texts. On top of that they added an equation editor for scientific texts, and spreadsheets that can be placed into the text similar to Ragtime. The whole program is a wonderful example of what MacApp can do in the hands of a competent person. Kamel Gaddas, one of the creators of Winsoft, shared his MacApp experience in his conference presentation.

Unfortunately, Wintext is copy-protected, and in a stupid way: even when installed on a hard disk it will ask for the master disk on each launch. Sorry to say, much as I'd like to buy the program, they won't get *my* money for a while.

[Note on the side - on copy protection, my favorite subject: while last year the standard answer was still 'well, in the US we might be able to do without copy protection, but for France there is no other way', this year we could hear 'well, in Europe we might be able to do without copy protection, but for the middle East markets there is no other way'... OK, so the targets are shifting. Let's hope they disappear].

AllPage was another multi-lingual word processing/ desktop publishing system by a company from Israel. Interesting feature: any selection, comprising text and graphics, can be copied as a picture and then pasted and re-sized.

Artificial intelligence (or whatever runs under this name) was represented in several exhibits: Procyon Research from Cambridge (UK) presented its Common Lisp implementation, which Expertelligence will distribute as a replacement of Exper Common Lisp. They distributed a very thorough introduction brochure and a demo diskette with a HyperCard stack. POP-11, the new AI programming language developed in Edinburgh and Sussex, was of course presented in its Mac implementation, AlphaPOP. Texas Instruments showed their Mac II-based AI workstation, microExplorer.

Mainstay, the makers of VIP, Think 'n Time, and other things, presented MarkUp, their new approach to text revision; sort of an electronic "red pencil". This is a system where a group of people can share a text document and make their comments on transparent "layers" which can be later selectively displayed on top of the document. So Joe can type a text, then have Debbie and Al revise it, later look at their comments on top of his text independently, change the text, make his own comments, etc. Comments can be entered into the text by typing or by using a set of graphics tools. Of course, color is supported. A pretty unique idea, and hopefully a success.

One particularly interesting contribution was an image processing application for microscopy Optimage, developed in Paris by university researchers and now being commercialized.

It is the first program that I'm aware of on the Mac that allows digital filtering, Fourier transforms, particle detection and counting, or size determination of objects in microscopic images.

Apple Expo 1988

The annual Paris Apple Expo was held during and after the conference. This time, the exposition hall, an old glass-steel construction, had been entirely transformed into a giant Mac II. 10-feet high floppies greeted the visitors in front of the building (how many gigabytes on that one?), and the - nonfunctional - 30- by 40- foot monitor on the outside served as a cloakroom. In the belly of the Mac II, the usual exposition come-and-go. Interesting the stand of Informix, in the form of a giant spaceship, where they introduced the long-awaited "hyper-spreadsheet" Wingz; this time they were announcing shipping in November. Vaporware special was the Gigatape, a 1.2 Gbyte SCSI tape backup which was supposed to work with DAT cassettes. When I asked for a demo, they said "we're still working on the driver, but it'll be ready Very Soon". Oh well.

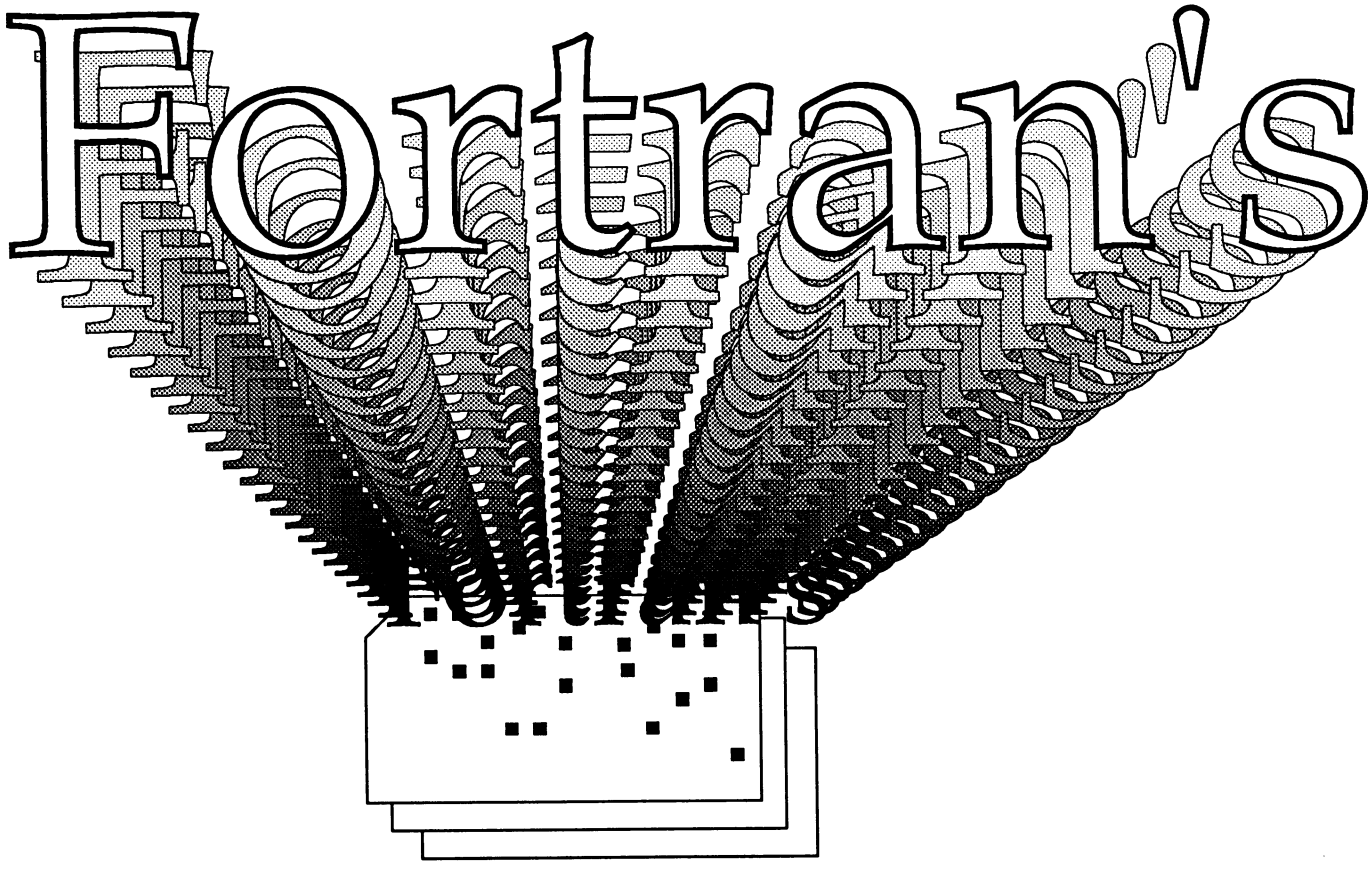
Apple occupied - naturally - a lot of floor space, and many of the more innovative applications were displayed on their stands; for instance a Labview-like data acquisition system, Personal Writer, a handwriting recognition system using a digitizer tablet, tons of educational HyperCard stacks, among them a system that will automatically find the best subway connection between points A and B in Paris - and also tell you how long it'll take (usually too long).

This year a contest was held; the Apple Trophy was awarded to seven programs selected by a jury from Apple France. 23 programs were on the short list, and I was promised to get a copy of that list so we could print it here; alas, it never arrived before our deadline (unless David receives a link from Apple France directly, in which case it might be printed after all). I'll just proudly mention that one of the members of our local Mac club, Yves Baulac, was among the prize winners with his geometry program, Cabri Geometre. The other six were Fileguard by Olivier Merenne, a hard disk protection/encryption facility that will protect your documents transparently; Hyperpage by Bernard Meunier, a page layout XCMD for HyperCard; Emploi du Temps, a planning utility by Jean-Luc Deleage; two medical utilities, Hypermed by Almanza and Thesaurus Urologie by Lardennois; and a graphing program, page de Graphe by Robert Elbeze.

Otherwise, the usual exhibitors showed their usual products (hello, Microsoft; bonjour, ACI). Most of the real new stuff had already been shown at the developers' fair. It was fun, anyway, to spend some time with friends at the MIDI stand.

Just as I write this, one day before the deadline, I've received the final release of Mach 2.14, where a lot of things have been added even with respect to the beta version. Finally, the editor allows you to PRINT your text as well!

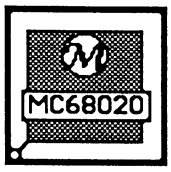
Next month it's back to Real Forth again. Till then.



World

© The Definitive MacTutor, Vol. 4

The Workstation Mac
Tips for Engineering Applications



Paul Zarchan
Cambridge, Mass.
MacTutor Vol. 4 No. 2

Introduction

With the advent of the Mac 2 and many third party Macintosh speed enhancers many engineers will now consider using the Macintosh for serious computation. Many of these engineers are currently using mainframe computers where the graphics environment is primitive or nonexistant and where CPU time is expensive and has little to do with turn around time. The purpose of this article is to give a few pointers in BASIC and FORTRAN, the languages with which engineers will initially feel most comfortable, on how to make effective use of the Macintosh for both graphics and speed.

Many engineers have heard that programming on the Mac is difficult and unlike mainframe programming. This is true if the end result is a commercial product. Making a program user friendly may cause simple code to grow by more than an order of magnitude. Using the various Macintosh toolbox routines requires the learning of a new language and a new way of thinking. However ordinary engineering applications are quite easy to program on the Mac - in fact even easier to program than on a mainframe or other micro - because of the user friendly environment, superior editing tools and virtually instantaneous feedback.

This article will show how engineers can modify ordinary programs, written in FORTRAN or BASIC, which generate numerical output and enhance these programs with the addition of a few lines of code so that they can serve as input drivers to professional commercial graphics software. In this way their numerical output can easily be converted to professional quality graphs and charts with little effort. They will discover, as I did, that instantaneous graphical feedback will not only help them convey their engineering message to others quickly but will also challenge them to modify their original program to get even more useful information. In addition, techniques for generating random numbers on the Mac will be discussed so that engineers can see how easy it is to perform monte carlo and statistical analysis. Finally, for those engineers wanting to run more complex problems on the Mac in BASIC, tips will be given on how to speed up program execution tiime with minimal changes to the program.

Plotting

The powerful graphic commands in MS BASIC and the access to the Macintosh toolbox in MacFortran permit the development of plotting packages in those respective languages. However, rather than re-inventing the wheel, it is better and much easier to use a very powerful commercial graphics package for engineers that already exists - Cricket Graph. The purpose of this section is to show the few lines of code are necessary to add to existing engineering programs so that the resultant output data,whether generated in BASIC or FORTRAN, can be transferred to Cricket Graph.

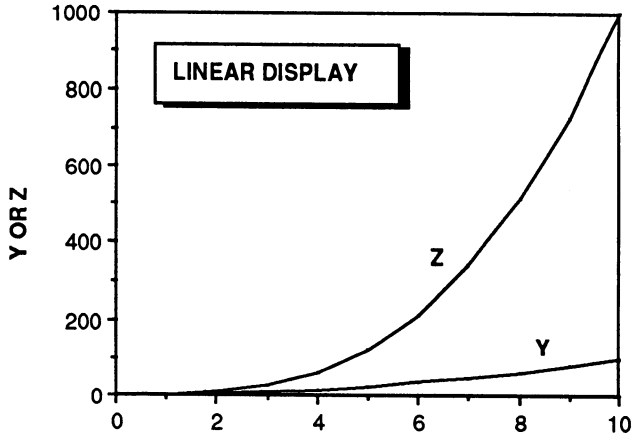
Cricket Graph requires it's numerical input to be in a tab delimited format. Unfortuneately MS BASIC writes data to a file in comma delimited format. The simple fix to the problem is to have BASIC write it's output to the screen for viewing purposes and to also write the output to the Macintosh clipboard for eventual pasting into Cricket Graph. Listing 1 presents a sample BASIC program which computes the expressions $y=i^2$ and $z=i^3$ for values of i ranging between 1 and 10. The resultant output is written to both the screen and clipboard.

```
OPEN "CLIP:"FOR OUTPUT AS #1
FOR I=1 TO 10
Y=I*I
Z=I^3
PRINT I,Y,Z
WRITE #1,I,Y,Z
NEXT I
CLOSE #1
```

Listing 1 - Writing Output to Clipboard in BASIC

The code in boldface can be added to any program so that output can be written to the clipboard in tab delimited format. The next step is to quit BASIC and then open Cricket Graph and finally to use the paste command to import the data. The resultant line plot, developed with Crickett Graph and slightly enhanced in MacDraw, is shown in Fig. 1.

Figure 1 - Using Cricket Graph to Plot BASIC Output



With Cricket Graph the ranges on the abscissa and ordinate can be changed quickly by double-clicking on the axis of interest. In addition, the BASIC data can be viewed in log-log form by simply choosing the log option in Cricket Graph. Grids can be added for additional readability. Figure 2 shows the same BASIC data viewed in gridded log-log format.

A FORTRAN program can also be modified so that its ouput can also be placed in a tabbed delimited format. This is easier to

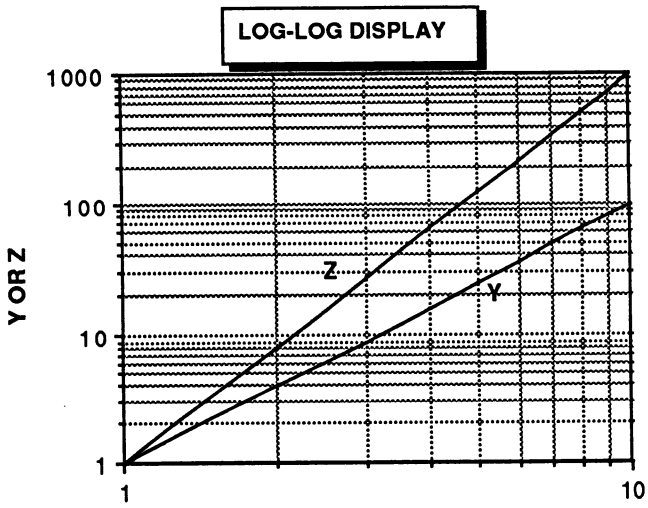
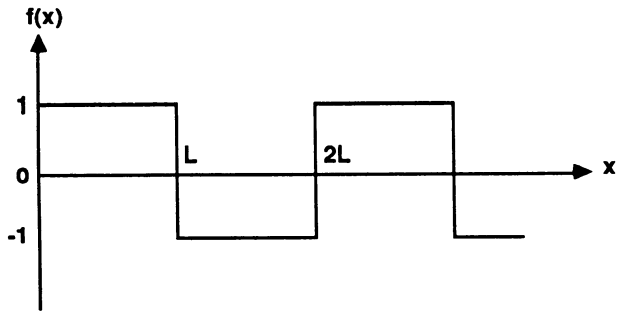


Figure 2 - BASIC Output Viewed in Log-Log Format

do in FORTRAN than in BASIC because it is possible to write data to a sequential file in tabbed delimited format. In order to illustrate the technique let's consider the Fourier Series representation of the periodic square wave shown in Fig. 3.

Figure 3 - Fourier Series Representation of Periodic Square Wave



$$f(x) = \frac{4}{\pi} \sum_{n=1,3,5,...}^{\infty} \frac{1}{n} \sin \frac{n \pi x}{L}$$

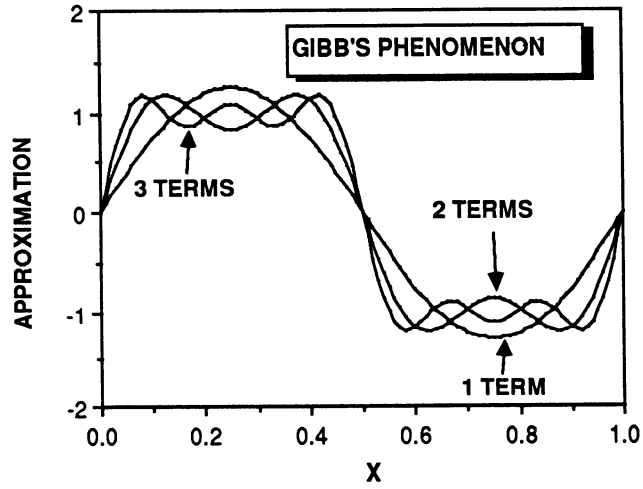
An infinite amount of terms would yield a perfect representation of the square wave whereas a finite number of terms would approximate the square wave. This effect is known as Gibbs phenomenon. Listing 2 presents a FORTRAN program which computes the first three terms of the periodic square wave Fourier Series and also writes the output to both the screen and also to a sequential file called "SAM".

```
      REAL L
      CHARACTER*1 TAB
      TAB=Z'09'
      OPEN(1,STATUS='NEW',FILE='SAM')
      L=.5
      PI=3.14159
      C1=4./PI
      DO 10 X=0,1,.01
      F1=C1*SIN(PI*X/L)
      F3=F1+C1*SIN(3.*PI*X/L)/3.
      F5=F3+C1*SIN(5.*PI*X/L)/5.
      WRITE(9,*)X,F1,F3,F5
      WRITE(1,97)X,TAB,F1,TAB,F3,TAB,F5
   97 FORMAT(F8.3,A1,F8.3,A1,F8.3,A1,F8.3)
   10 CONTINUE
      CLOSE(1)
      PAUSE
      END
```

Listing 2 - Writing Tabbed Delimited Output to File in FORTRAN

The boldfaced statements in the above listing can be added to any FORTRAN program to write the output data, in a tabbed delimited format, to a sequential file. The sequential file can then be directly opened by Cricket Graph for plotting. For this example the resultant Cricket Graph output is displayed in line chart form in Fig. 4.

Figure 4 - FORTRAN Output Displayed in Cricket Graph



Random Numbers

Random numbers play an important role in engineering applications. They form the cornerstone of monte carlo and statistical analysis. In BASIC, uniformly distributed random numbers are available with the **RND** statement. Other random distributions can be constructed from the uniform distribution. For example, since the **RND** statement creates a uniformly distributed number between 0 and 1 with variance 1/12 and mean .5, a gaussian distributed random variable with variance 1 and zero mean can be constructed, according to the central limit theorem, by adding 12 uniformly distributed numbers together (12 times 1/12 is one) and subtracting 6. Therefore the following BASIC statement generates an approximate Gaussian distributed random variable with zero mean and unity standard deviation.

```
X=RND+RND+RND+RND+RND+RND+RND+RND+RND+RND+RND+RND-6
```

The distribution is approximately Gaussian since the maximum and minimum values of the distribution are ±6 rather than ±• . If the approximate Gaussian number generator is in a program inner loop and computer running time needs to be enhanced, fewer terms can be used in the above expression to speed up the program. For example, a six term approximate Gaussian number generator with unity variance and zero mean can be constructed from

```
X=1.414*(RND+RND+RND+RND+RND+RND-3)
```

In this case the maximum and minimum values of the approximate distribution are ±3 rather than ±• .

© The Definitive MacTutor, Vol. 4

FORTRAN, unlike BASIC does not provide a uniform random number generator. However an efficient random number generator can be constructed in FORTRAN by making one toolbox call. TOOLBX(RANDOM) generates a uniformly distributed integer between ±32768 quickly. Listing 3 presents a FORTRAN program which, by using the above toolbox call, generates approximate gaussian random numbers and then puts the numbers into bins so that a probability density function can also be constructed.

```
      INTEGER BIN,RANDOM,SUM
      INTEGER*4 TOOLBX
      DIMENSION H(10000),X(10000)
      PARAMETER (RANDOM=Z'86140000')
      CHARACTER*1 TAB
      TAB=Z'09'
      OPEN(2,STATUS='NEW',FILE='SAM')
      XMAX=6.
      XMIN=-6.
      RANGE=XMAX-XMIN
      TMP=1/SQRT(6.28)
      N=1000
      BIN=50
      DO 10 I=1,N
      SUM=0
      DO 14 J=1,12
      IRAN=TOOLBX(RANDOM)
      SUM=SUM+IRAN
   14 CONTINUE
      X(I)=SUM/65536.
   10 CONTINUE
      DO 20 I=1,BIN
      H(I)=0
   20 CONTINUE
      DO 30 I=1,N
      K=INT(((X(I)-XMIN)/RANGE)*BIN)+.99
      IF( K<1)K=1
      IF(K>BIN)K=BIN
      H(K)=H(K)+1
   30 CONTINUE
      DO 40 K=1,BIN
      PDF=(H(K)/N)*BIN/RANGE
      AB=XMIN+K*RANGE/BIN
      TH=TMP*EXP(-AB*AB/2.)
      WRITE(9,*)AB,PDF,TH
      WRITE(2,97)AB,TAB,PDF,TAB,TH
   97 FORMAT(F8.3,A1,F8.3,A1,F8.3)
   40 CONTINUE
      CLOSE(2)
      PAUSE
      END
```

Listing 3 - FORTRAN Program Illustrating Random Number Generation

The highlighted terms are those statements which are required by any FORTRAN program to generate uniformly distributed or gaussian distributed random numbers. Here X(I) is the resultant gaussian distributed number. In this particular program 1000 random numbers are calculated. The accuracy of the random number generator is illustrated by viewing the program output, shown in Fig. 5, which compares the sampled probability density function to the theoretical bell-shaped curve.

Speeding Up BASIC Programs

In a previous issue MacTutor issue[1] it was shown how the transient response of a sixth order Butterworth filter could be simulated in FORTRAN using the second order Runge Kutta method for integrating the six differential equations. A block

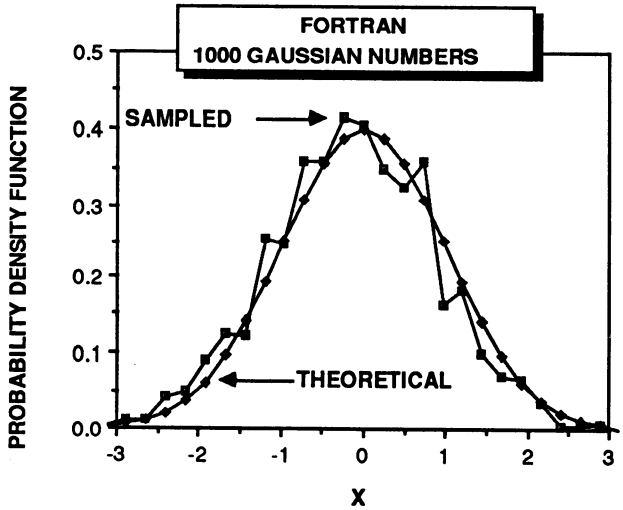


Figure 5 - FORTRAN Generates Gaussian Distributed Random Numbers Accurately

diagram of the Butterworth filter is repeated here for convenience in Fig. 6. Here the "1/s" terms represent integrations.

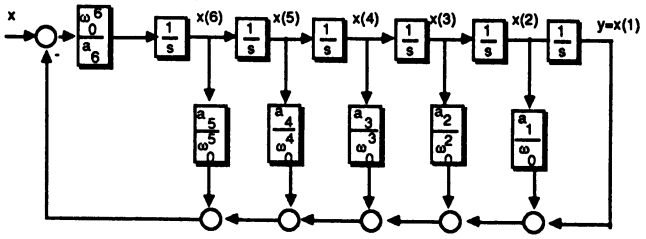


Figure 6 - Sixth Order Butterworth Filter

The resultant six first order differential equations, which can be obtained from Fig. 6, are:

$$\dot{x}(1) = x(2)$$

$$\dot{x}(2) = x(3)$$

$$\dot{x}(3) = x(4)$$

$$\dot{x}(4) = x(5)$$

$$\dot{x}(5) = x(6)$$

$$\dot{x}(6) = \frac{\omega_0^6}{a_6}\left[x - x(1) - \frac{a_1 x(2)}{\omega_0} - \frac{a_2 x(3)}{\omega_0^2} - \frac{a_3 x(4)}{\omega_0^3} - \frac{a_4 x(5)}{\omega_0^4} - \frac{a_5 x(6)}{\omega_0^5}\right]$$

A BASIC program which can integrate the above differential equations is shown in Listing 4.

```
DIM X(6),XOLD(6),XD(6)
START=TIMER
B1=3.86:B2=7.46:B3=9.13:B4=7.46:B5=3.86:B6=1:W0=50
XIN=1
ORDER=6
FOR I=1 TO ORDER
X(I)=0
NEXT I
T=0
H=.0005
S=0
BEGIN:
IF T>=.5 GOTO FINISH
```

```
    S=S+H
    FOR I=1 TO ORDER
    XOLD(I)=X(I)
    NEXT I
    GOSUB INTEG
    FOR I=1 TO ORDER
    X(I)=X(I)+H*XD(I)
    NEXT I
    T=T+H
    GOSUB INTEG
    FOR I=1 TO ORDER
    X(I)=.5*(XOLD(I)+X(I)+H*XD(I))
    NEXT I
    IF S>.004999 THEN
    S=0
    PRINT T,X(1)
    END IF
    GOTO BEGIN
    FINISH:
    PRINT TIMER-START
    WHILE INKEY$="":WEND
    INTEG:
    XD(1)=X(2)
    XD(2)=X(3)
    XD(3)=X(4)
    XD(4)=X(5)
    XD(5)=X(6)
    XD(6)=(W0^6)*(XIN-B5*X(6)/(W0^5)-B4*X(5)/(W0^4)-B3*X(4)/
(W0^3)-B2*X(3)/(W0^2)-B1*X(2)/W0-X(1))/B6
    RETURN
    END
```

Listing 4 - BASIC Simulation of Butterworth Filter

We can see from the listing that the integration step size, H, is .0005 sec. The running time for this program in interpretive MS BASIC is 229 sec on a Macintosh. Halving the step size or doubling the simulation time will double the running time. The program can be speeded up by using the MS BASIC Compiler and the resultant running time will decrease to 56 sec. Other methods can also be used to further enhance speed. For example the compiler and interpreter assume that all arrays are dynamic. However we can tell the compiler, but not the interpreter, that the arrays are static (as long as we aren't going to redimension statements during the program) and significantly decrease the running times. In addition, making loop parameters integers, as is commonly done in FORTRAN, rather than default single precision real numbers will also enhance running time - especially with the compiler. Converting real numbers to integers can easily be accomplished in BASIC by adding a percent sign to the variable. Finally eliminating repeated calculations of constants from loops will also speed up the program. For example, the derivative XD(6) has many such constant computations which can easily be moved to the beginning of the program without destroying the readability of the code. The faster running BASIC program is shown in Listing 5.

```
    DIM X(6),XOLD(6),XD(6)
    START=TIMER
    B1=3.86:B2=7.46:B3=9.13:B4=7.46:B5=3.86:B6=1:W0=50
    XIN=1
    ORDER%=6
    W02=W0*W0
    W03=W02*W0
    W04=W03*W0
    W05=W04*W0
    W06=W05*W0
    FOR I%=1 TO ORDER%
    X(I%)=0
    NEXT I%
    T=0
    H=.0005
    S=0
    BEGIN:
    IF T>=.5 GOTO FINISH
    S=S+H
    FOR I%=1 TO ORDER%
    XOLD(I%)=X(I%)
    NEXT I%
    GOSUB INTEG
    FOR I%=1 TO ORDER%
    X(I%)=X(I%)+H*XD(I%)
    NEXT I%
    T=T+H
    GOSUB INTEG
    FOR I%=1 TO ORDER%
    X(I%)=.5*(XOLD(I%)+X(I%)+H*XD(I%))
    NEXT I%
    IF S>.004999 THEN
    S=0
    PRINT T,X(1)
    END IF
    GOTO BEGIN
    FINISH:
    PRINT TIMER-START
    WHILE INKEY$="":WEND
    INTEG:
    XD(1)=X(2)
    XD(2)=X(3)
    XD(3)=X(4)
    XD(4)=X(5)
    XD(5)=X(6)
    XD(6)=W06*(XIN-B5*X(6)/W05-B4*X(5)/W04-B3*X(4)/W03-
B2*X(3)/W02-B1*X(2)/W0-X(1))/B6
    RETURN
    END
```

Listing 6 - Faster BASIC Program

The terms that were changed to speed up the program are highlighted. Table 1 documents the speedup enhancements.

Condition	Interpreter	Compiler
Nominal	229 sec	56 sec
Make Arrays Static	229 sec	42 sec
Use Integers in Loops	211 sec	26 sec
Move Constant Computations Out of Loops	120 sec	22 sec

Table 1 - Improving Speed of BASIC

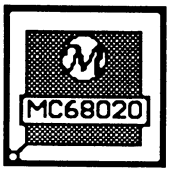
We can see from Table 1 that there is a big payoff in using static arrays and integers when using the compiler. On the other hand, moving constant computations out of loops is more effective, in this case, when using the interpreter. For this example we can see that the net speed-up in being careful and using the compiler, rather than the interpreter, is more than a factor of 10 with only minor modifications to the original code. The best policy is to first get the programmed debugged and running using the interpreter because errors can be found and corrections can be made virtually instaneously. The tips for speed enhancement can then be applied when the program is checked out and is ready to be compiled.

References

1. Zarchan, P. "New Mac Workstation Potential", MacTutor, March 1987, pp. 15-21.

The Workstation Mac

MatLab Comes to the Mac II



Paul Zarchan
Cambridge, Mass
MacTutor Vol. 4 No. 5

Introduction

MATLAB is an interactive program to help with scientific and engineering numeric calculations. Although it was originally written in FORTRAN for mainframe computers, a second generation MATLAB was rewritten for smaller computers such as the Sun Workstation and the VAX computer. In the last few years a special version of MATLAB was developed to run on MS DOS compatable PC's with math coprocessors. Typical uses for MATLAB are general purpose numeric computation, algorithm prototyping and solving special purpose problems with matrix formulations that arise in disciplines such as automatic control theory, statistics and digital signal processing. MATLAB is now available for the Mac II or for any Mac souped up with a 68020 accelerator board and a 68881 math coprocessor. For engineers this lends legitimacy to the claim that the Mac II is actually a workstation.

Unlike FORTRAN on the MAC, MATLAB is not really designed to make executable code (i.e. "double clickable" applications). As with interpretive BASIC, the MATLAB application is required on the disk to execute any file previously generated. Unlike BASIC, MATLAB is very fast because of the highly efficient algorithms. MATLAB allows engineers to solve problems almost as they are written mathematically and in addition allows easy access to libraries of some very powerful software. Some have called MATLAB "APL without the pain". The Mac version of MATLAB takes full advantage of the Mac interface. This also means the MATLAB output can easily be incorporated into reports and presentations making MacMATLAB far more useful than the DOS version.

Matrix Examples

Matricies can be entered either in the MATLAB editor or by the carrot prompt by using brackets to define the matrix and semicolons to separate matrix rows. For example, by entering

```
»A=[1 2 3;4 5 6;7 8 0]
```

we get the MATLAB output on screen as

```
A =
   1   2   3
   4   5   6
   7   8   0
```

If we want to find the transpose of matrix A we simply type at the prompt

```
»B=A'
```

where the ' indicates transposition The resultant screen output is

```
B =
   1   4   7
   2   5   8
   3   6   0
```

Taking the matrix inverse of A is simple as typing

```
»inv(A)
```

which yields the screen output (nearly instantaneously)

```
ans =
  -1.7778   0.8889  -0.1111
   1.5556  -0.7778   0.2222
  -0.1111   0.2222  -0.1111
```

Many other matrix and linear algebra functions are available with MATLAB. Of course, many engineers have these functions built into their FORTRAN libraries - so why use MATLAB? Let's do more examples to see why!

Polynomial Example

Suppose we had two polynomials and we wanted to multiply them and find the roots of the resultant expression. If the two polynomials were

x^2+3x+2 and x^2+x-12

we would define the polynomials in MATLAB by typing

```
»A=[1 3 2];
»B=[1 1 -12];
```

where the polynomials are expressed as vectors where the vector elements represent the polynomial coefficients in descending order. We can multiply the 2 polynomials algebraically in MATLAB by using the convolution function or

```
»C=conv(A,B)
```

which yields

```
»C =
   1   4  -7  -34  -24
```

The screen output displays the coefficients of the resultant polynomial in descending order. This means that the product of the 2 polynomials is given by

$x^4+4x^3-7x^2-34x-24$

To find the roots of the resultant polynomial we simply type

```
»roots(C)
```

which yields the 4 roots as

```
ans =
   3.0000
  -4.0000
  -2.0000
  -1.0000
```

Still not impressed with MATLAB? Let's look at more examples.

Plotting

Unlike FORTRAN, plotting functions come built into MATLAB. If we wanted to plot

$y=\sin(x)$ and $z=.01x^2$ from -15 to 15 in increments of .05 only the following 4 MATLAB statements would be required

```
»x=-15:.05:15;
»y=sin(x);
»z=x.^2/100;
»plot(x,y,x,z)
```

The unenhanced screen output for such a set of commands appears in Fig. 1. Of course since MATLAB grpahical output is in the PICT format it can be copied to the clipboard and pasted into any Macintosh application.

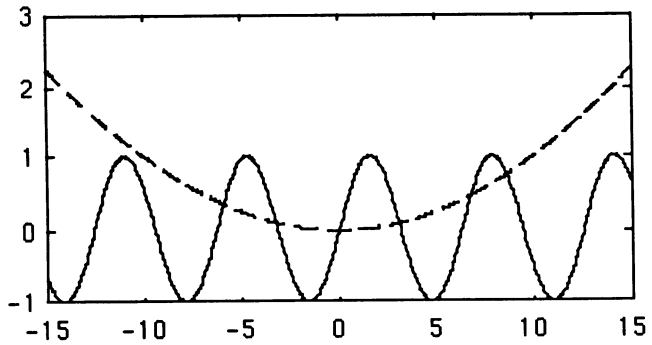


Figure 1 - Plotting With MATLAB is Easy

The graph can be embellished within the MATLAB environment (with features similar to Cricket Graph) or copied onto the clipboard and pasted into MacDraw for further enhancement.

If we wanted to show that a 5 term Fourier series expansion for a square wave is accurate (Gibb's effect) we only need the following 3 lines of MATLAB code.

```
»t = 0:.1:10;
»y = sin(t) + sin(3*t)/3 + sin(5*t)/5 + sin(7*t)/7 + sin(9*t)/9;
»plot(t,y)
```

The resultant unenhanced MATLAB output is shown below.

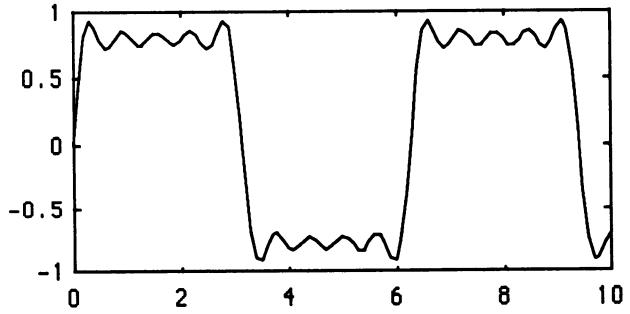


Figure 2 - MATLAB Output For Gibb's Effect

Still not impressed? Suppose we wanted to evaluate the function

$$z = x\,e^{(-x^2-y^2)}$$

over the range
$-2 \le x \le 2,\ -2 \le y \le 3$

All we have to do is use the following 3 MATLAB statements to get a 3 dimensional plot (with all the hidden line algorithms transparent to the user) of the function.

```
»[x,y]=meshdom(-2:.2:2,-2:.2:3);
»z=x.*exp(-x.^2-y.^2);
»mesh(z)
```

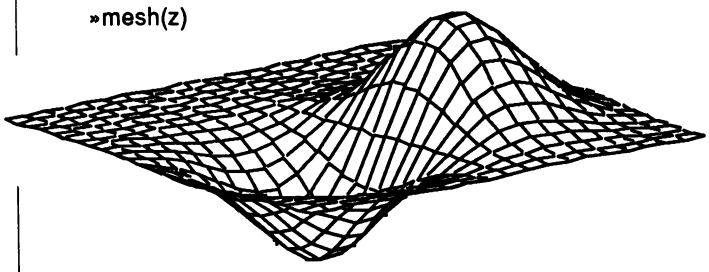


Figure 3 - MATLAB 3 Dimensional Plot

Incidentally, the plot took only a few seconds to generate. Try doing that in FORTRAN!

Signal Processing

Noisy signals can be analyzed with MATLAB. For example, we can make 750 gaussian distributed random numbers with zero mean and unity standard deviation with the statements

```
»rand('normal')
»y=rand(750,1);
```

We can then analyze and display the frequency content of the 750 random numbers with a MATLAB histogram statement or

```
»[n,x]=hist(y);
»plot(x,n,'+')
```

As expected, the MATLAB output, shown in Fig. 4, demonstrates that the 750 random numbers follow the standard "bell-shaped" curve or well known gaussian distribution.

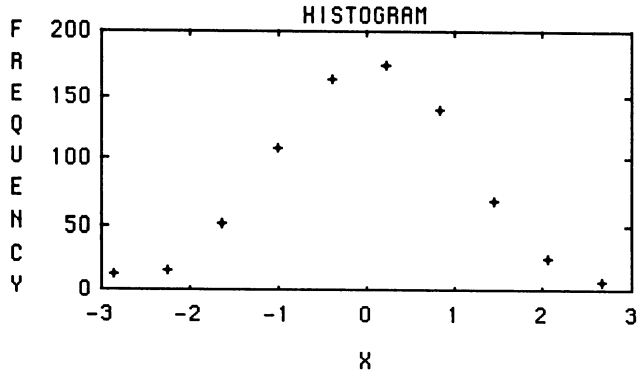


Figure 4 - MATLAB Can Make Histograms

As was mentioned in the introduction, MATLAB comes with very powerful algorithms. The next example shows a simple use of the FFT (Fast Fourier Transform) function. A common use of the FFT is to find the frequency components buried in a noisy time domain signal. Consider data sampled at 1000 Hz. We can form a signal containing 50 Hz and 120 Hz and corrupt it with gaussian noise (zero mean and unity standard deviation) with the following 5 MATLAB statements.

```
»t=0:.001:.5;
»x=sin(2*pi*50*t)+sin(2*pi*120*t);
»rand('normal')
»y=x+2*rand(t);
»plot(y(1:50))
```

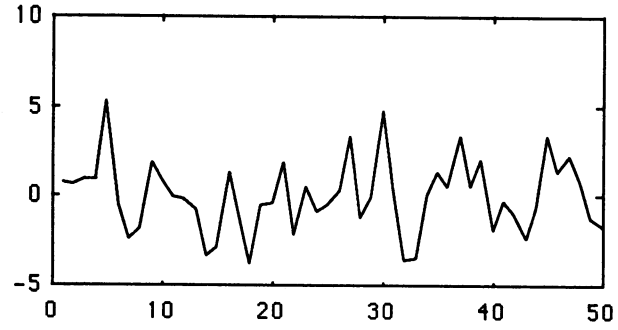


Figure 5 - Noisy Signal in Time Domain

It is difficult, if not impossible, to identify the frequency components by looking at the original signal in the time domain. Converting to the frequency domain, the discrete Fourier transform of the noisy signal y is found by taking the FFT. In MATLAB code this means

```
»Y=fft(y);
```

The power spectral density, a measurement of the energy at various frequencies, is:

```
»Pyy=Y.*conj(Y);
```

We can plot the power spectral density by forming a frequency axis for the first 256 points (the other 256 points are symmetric) with the MATLAB statements

```
»f=1000*(0:255)/512;
»plot(f,Pyy(1:256))
```

From the resultant output in Fig. 6 we can clearly see the frequency components at 50 Hz and 120 Hz of the signal.

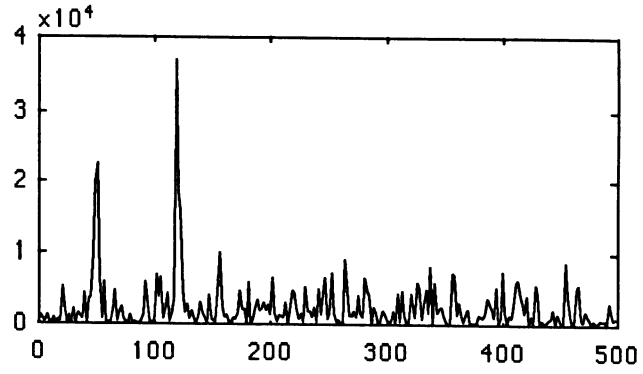


Figure 6 - Frequency Domain Output

Control Analysis

MATLAB can also be used in the analysis of control systems from both a modern and classical point of view. We can find and plot the root locus of the transfer function

$$H(s) = \frac{.2\,s^2 + .3\,s + 1}{(s^2 + .4\,s + 1)(s + .5)}$$

for gains 0 to 10 in steps of .5 with the following MATLAB statements

```
»num = [.2  .3  1];
»den1 = [1 .4 1];
»den2 = [1 .5];
»den = conv(den1,den2);
```

```
»k = 0:.5:10;
»r = rlocus(num,den,k);
»plot(r,'x'), title('Root-locus'),xlabel('Real
part'),ylabel('Imag part')
```

The resultant root locus is shown in Fig. 7.

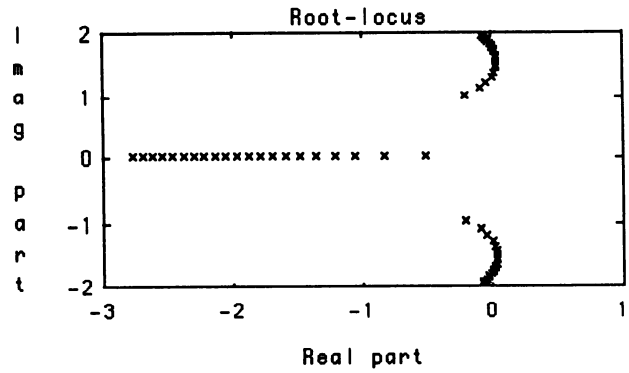


Figure 7 - Root Locus Example With MATLAB

We could find the step response for the same system by typing the additional statements

```
»[a,b,c,d] = tf2ss(num,den)
»t = 0:.3:15;
»y = step(a,b,c,d,1,t);
»plot(t,y),title('Step response'),xlabel('time(sec)')
```

yielding the output of Fig. 8

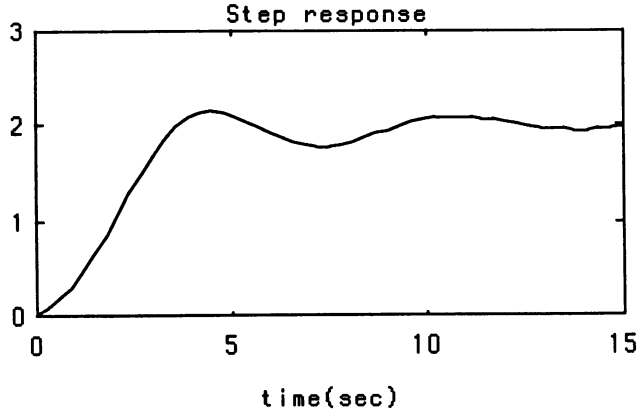
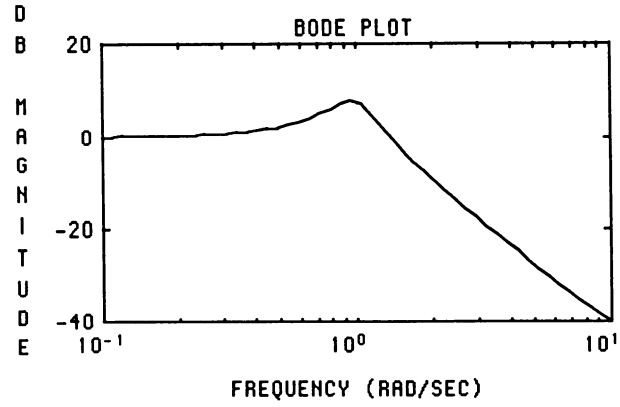


Figure 8 - Step Response Output

Another example of the control system analysis capability of MATLAB consider a second order quadratic transfer function with natural frequency 1 rad/sec and damping of .2. The MATLAB program required to display the magnitude characteristics is shown below and the resultant output is displayed in Fig. 9.

```
»[a,b,c,d]=ord2(1,.2);
»w=logspace(-1,1);
»[mag,phase]=bode(a,b,c,d,1,w);
»mag1=20*log10(mag);
»semilogx(w,mag1)
```

Figure 9 - Sample Magnitude Bode Plot For Quadratic Transfer Function



More Details

If you already used MATLAB in the DOS world and have a collection of many files there is no problem in porting them to the MAC. Just use a file transfer program (I used PC to MAC and Back) and the conversion routine (going from ASCII files to .m files) provided by MATLAB. The danger in doing this is that the DOS version of MATLAB will seem very primitive once you have used the MAC version. Incidentally the MAC version of MATLAB runs about 5 times faster than the IBM/AT version and at about the same speed as the Sun-3 Workstation version.

Macintosh MATLAB was developed by *The Math Works, Inc.* [20 N. Main St., Suite 250, Sherborn, Mass. 01770, (617)653-1415]. The software, including the signal processing toolbox comes on a single 800k disk (unprotected) with a list price of $895. The control and ientification toolboxes are sold separately - each with a list price of $495. Although this software is expensive by MAC word processing and spreadsheet standards - it is inexpensive by engineering software standards.

Fortran's World

Fortran Printing Interface

Jay Lieske
La Canada, CA
MacTutor Vol. 4 No. 8

In Absoft FORTRAN version 2.3 one usually must employ Absoft's special routine called PrPort.sub in order to use the Printing Manager on the Macintosh. PrPort.sub employs the old RAM-based printing manager, but recent trends suggest that it would be better to use the printing traps accessible through "_PrGlue" at $A8FD since these routines are now in ROM and contain special routines such as PrGeneral that are unavailable in the RAM-based printing manager. In addition, PrPort.sub, which is written in assembly language, is limited to the hard-coded routines which existed at the time Absoft developed the routine and is thus not extendible. Since usage of PrPort.sub will undoubtedly limit the development of FORTRAN programs on the Macintosh, it is of interest to see how one might use Absoft's Toolbx.sub in order to implement the new ROM-based printing manager traps.

The Toolbx.sub routine from Absoft employs an encoded 'trap dispatch' LongInt to describe the toolbox or OS trap number, as well as to describe the parameters required for the toolbox or OS calls. The new printing traps accessible through the _PrGlue trap at $A8FD require that a special 'routine selector' be pushed onto the stack prior to the $A8FD trap instruction. Hence, if we are to succeed in using the print traps, we must be able to insert this routine selector number onto the stack.

By studying and disassembling Absoft's Toolbx.sub it was found that the routine selector parameter could be put into the Toolbx.sub call as the final parameter (i.e. the parameter just before the final closing parenthesis) and that it would be pushed onto the stack just before Toolbx.sub executed the trap number. In order to instruct Toolbx.sub that there was another parameter for it to pass, however, the 'trap dispatch' code had to be modified. That is where the disassembly of Toolbx.sub became important because current Absoft documentation is not always correct on this matter.

We have developed the include files 'prtrap.inc' and 'PrGenDefs.inc' which can be used with Absoft Fortran's Toolbx.sub in order to implement the print traps. A similar method could be employed for any other new traps (or traps and additional routine selectors') that Apple develops. The file 'prtrap.inc' contains the encoded trap dispatch number and the routine selector for all the new print traps documented in Inside Macintosh Volume V. The routine selector for PrGeneral, which inadvertently was left out of Inside Mac, was found in Lightspeed Pascal's equates as $70070480. We also include a file 'PrGenDefs.inc' which contains the equates required for use of the PrGeneral gateway, potentially a very useful new trap which enables one to check and to set the resolution of the printer in use.

To demonstrate how the Toolbx.sub traps are implemented in a program, first consider the Pascal calling-sequences as documented in Inside Mac. Suppose that a trap 'PrRoutine' is documented by Apple as being implemented like this in Pascal:

PrRoutine(Arg1, Arg2,...,Argn);

then the Toolbx.sub routine in Fortran would be employed as follows:

call Toolbx(PrRoutineCode, Arg1, Arg2,...,Argn, xPrRoutineCode) if the toolbox trap is a procedure. If the toolbox call is a function then the Pascal version would be

Result := PrRoutine(Arg1, Arg2,...,Argn);

while the Toolbx.sub Fortran version would be

Result = Toolbx(PrRoutine,Arg1, Arg2,...,Argn,xPrRoutine).

The analogous calls using prport.sub for a subroutine would be: call PrPort(Code, Arg1, Arg2,..., Argn). Programs which currently employ PrPort could readily be changed simply by replacing PrPort with Toolbx and appending the 'xPrRoutine' parameter.

An example program 'demotrap.for' is included for implementing the new print manager trap-based calls. The program does not actually do any printing, but shows how one can access the print record and query the printer for variable resolutions. Just use the Chooser to select your 'printer' and then the usual dialogs will come up on screen to demonstrate that everything works as advertised. A colleague has employed these new traps in a large plotting package which was ported over to the Mac from a mainframe computer and the results are quite satisfactory.

Listing One: Demonstration Program

```
      program demotrap
  C     See if can use 'Printer traps' via $A8FD PrGlue and
TOOLBX.sub
  C  A simple sample program demonstrating its capability
for Absoft Fortran 2.3.
  C  This program does not actually do any printing.
  C  J. Lieske 12/87

  C * * * * * * * * * *
  * FUNCTION PTR (ANYTYPE) : PTR;                ; A 'funny'
trap.
       INTEGER PTR                              ! from
Absoft's MISC.inc
       PARAMETER (PTR=Z'C0000000')

       integer NEWHANDLE
  ! from Absoft's MEMORY.inc
       parameter (NEWHANDLE=Z'122000A8')
       integer iPrintSize
  ! The Print record size.[120 bytes]
       parameter (iPrintSize=120)
  ! from Absoft's PRDEFS.inc
```

© The Definitive MacTutor, Vol. 4

```
C * * * * * * * * * *
C   GRXMXØ : default x size (pixels).  These will be taken
from the print record
C   GRYMXØ : default y size (pixels)
C   GRXPIN : pixels per inch in x
C   GRYPIN : pixels per inch in y
      REAL  GRXPIN, GRYPIN
      INTEGER   GRXMXØ, GRYMXØ
      INTEGER*4 TOOLBX
! MAC TOOLBOX INTERFACE

x     integer*4 debugger
! just in case you want to use MacsBug
x     parameter (debugger=z'9FF00000')
x     call toolbx(debugger)
! put the call where it's needed
*
* INITIALIZE THE PRINT RECORD FOR THE MACINTOSH
*
      integer*2 iversion
      integer*2 ernum

 * PRINT MANAGER FUNCTION DEFINITIONS for toolbox-based
traps
      include prtrap.inc
      include PrGenDefs.inc

      INTEGER PRRECHDL
            ! HANDLE TO THE PRINT RECORD.
      INTEGER POINT
            ! DEREFERENCED HANDLE
      LOGICAL OK

      write(*,*) 'This program demonstrates printer traps
in Fortran.'
      write(*,*) 'It won''t actually print anything, but
shows how to'
      write(*,*) 'access the printer traps in Absoft
Fortran.'
      write(*,*) 'It will write the data out to
''output.fil'''
      write(*,*) 'so you can read it.'

      call toolbx(propen, xpropen)  ! PrOpen

      PRRECHDL = TOOLBX(NEWHANDLE, IPRINTSIZE)  ! GET A
PRINT RECORD HANDLE.
      call toolbx(printdefault,PRRECHDL,xprintdefault)
! get defaults

C  get printer version number
      iversion = toolbx(prdrvrvers, xprdrvrvers )
      write (*,*) 'iversion = ', iversion
      open(15,file='output.fil', status='NEW')
      write (15,*) 'iversion = ', iversion

C  see if PrGeneral will work
      gOpCode = 4       ! set iOpCode
C-use PTR to get address of data record
      pGetRslData = toolbx(PTR,GetRslDataRec)
      call toolbx(PrGeneral, pGetRslData, xPrGeneral)
C  should now check for PrError also
      ernum = toolbx(PrError, xPrError)
      write(*,*) 'PrError number = ', ernum
      write(15,*) 'PrError number = ', ernum

      write(*,*) 'Interpreted record'
      write(*,*) 'iOpCode= ',gOpCode
      write(*,*) 'iError= ',gError
      write(*,*) 'lReserved= ',gReserved
      write(*,*) 'iRgType= ',iRgType
      write(*,*) 'XRslRg= ',XRslRg
      write(*,*) 'YRslRg= ',YRslRg
      write(*,*) 'iRslRecCnt= ',iRslRecCnt
      write(*,*) 'rgRslRec= ',((rgRslRec(i,j), i=1,2),
j=1,iRslRecCnt)

      write(15,*) 'Interpreted record'
      write(15,*) 'iOpCode= ',gOpCode
      write(15,*) 'iError= ',gError
      write(15,*) 'lReserved= ',gReserved
      write(15,*) 'iRgType= ',iRgType
      write(15,*) 'XRslRg= ',XRslRg
      write(15,*) 'YRslRg= ',YRslRg
      write(15,*) 'iRslRecCnt= ',iRslRecCnt
      write(15,*) 'rgRslRec= ',((rgRslRec(i,j), i=1,2),
j=1,iRslRecCnt)

  C now try to set it to 300 dpi
      pOpCode = 5       ! set iOpCode for SetRsl
      hPrint = PRRECHDL ! handle to print record
      iXRsl = 300
      iYRsl = 300
      pSetRsl = toolbx(PTR,SetRslRec)
      call toolbx(PrGeneral, pSetRsl, xPrGeneral)

      OK = toolbx(prstldialog,PRRECHDL,xprstldialog)  !
std dialog

      OK=toolbx(prjobdialog,PRRECHDL,xprjobdialog)    !
PrJobDialog
      if (.not. OK) STOP
  *   NOW READ PLOT DIMENSION VALUES WITH THOSE FROM THE
PRINTER INFO RECORD.
  *   PRRECHDL IS A HANDLE TO THE PRINTER RECORD WHICH IS
120 BYTES
  *   THE VALUES WE CARE ABOUT ARE THE VERTICAL AND
HORIZONTAL RESOLUTION
  *   AND THE PAGE RECTANGLE WHICH ARE AT OFFSETS 4,6, AND
8 INTO THE RECORD

      POINT=LONG(PRRECHDL)
      GRXMXØ = WORD(POINT+14)
      GRYMXØ = WORD(POINT+12)
      GRXPIN = FLOAT(WORD(POINT+6))
      GRYPIN = FLOAT(WORD(POINT+4))
      write(*,*)  'resolutions',
GRXMXØ,GRYMXØ,GRXPIN,GRYPIN
      write(15,*)
'resolutions',GRXMXØ,GRYMXØ,GRXPIN,GRYPIN
      close(15)


  *
  *   CLOSE UP THE PRINTER
  *
      call toolbx(prclose,xprclose)       ! Close the
printing grafport

      pause 'Press return to exit'
      stop
      end
```

Listing Two: PrTrap.inc

```
  * File: prtrap.inc      29 Dec 1987       JHL
  *     A replacement for Absoft's prport.inc and prport.sub
.
  *     This implements traps for Fortran so that it can be
used with toolbx.sub .
  *     Called together by J. Lieske, Jet Propulsion
Laboratory, based upon
  *     Inside Mac V and Lightspeed Pascal equates.
  *
  *     The general calling sequence for procedures is
  *           CALL TOOLBX(PrRoutine, Args, xPrRoutine)
  *     while that for a function is
  *           TOOLBX(PrRoutine, Args, xPrRoutine) .
  *     In the above parameters, PrRoutine is the encoded
PrGlue (Printer Trap
  *     $A8FD) parameter and xPrRoutine is the 'call number'
as specified in
  *     Inside Mac V Chapter 22.  The 'Args' are the
sequence of arguments to
  *     the procedure or function as specified by Inside
Mac.
  *           Basically, the xPrRoutine 'call number' is
placed on the stack
  *     just ahead of the printer trap number ($A8FD) and
the Fortran PrRoutine
  *     parameter tells Fortran's TOOLBX how to interpret
the 'Args' and
  *     'xPrRoutine' values.
  *           For example, the routine PrintDefault is
called from
  *     Fortran as follows:  CALL TOOLBX( PRINTDEFAULT,
hPrint, xPRINTDEFAULT) .
  *
  * * * * * * * * * * * * * * * * * * * * * *
  *     Modification history:
  *     12/87 JHL   Developed basic parameters from Absoft's
documentation
  *                 of toolbx.sub and Apple's Inside Mac V
and Lightspeed
  *                 Pascal.
  * * * * * * * * * * * * * * * * * * * * * *
  * PROCEDURE PrOpen;
        INTEGER*4 PROPEN, xPROPEN
        PARAMETER (PROPEN=Z'8FD10000', xPROPEN=Z'C8000000')

  * PROCEDURE PrClose;
        INTEGER*4 PRCLOSE, xPRCLOSE
        PARAMETER (PRCLOSE=Z'8FD10000',
xPRCLOSE=Z'D0000000')

  * PROCEDURE PrintDefault (hPrint: THPrint);
        INTEGER*4 PRINTDEFAULT, xPRINTDEFAULT
        PARAMETER (PRINTDEFAULT=Z'8FD12000',
xPRINTDEFAULT=Z'20040480')

  * FUNCTION PrValidate (hPrint: THPrint) : BOOLEAN;
        INTEGER*4 PRVALIDATE, xPRVALIDATE
        PARAMETER (PRVALIDATE=Z'8FDD2000',
xPRVALIDATE=Z'52040498')

  * FUNCTION PrStlDialog (hPrint: THPrint) : BOOLEAN;
        INTEGER*4 PRSTLDIALOG, xPRSTLDIALOG
        PARAMETER (PRSTLDIALOG=Z'8FDD2000',
xPRSTLDIALOG=Z'2A040484')

  * FUNCTION PrJobDialog (hPrint: THPrint) : BOOLEAN;
        INTEGER*4 PRJOBDIALOG, xPRJOBDIALOG
        PARAMETER (PRJOBDIALOG=Z'8FDD2000',
xPRJOBDIALOG=Z'32040488')

  * PROCEDURE PrJobMerge (hPrintSrc, hPrintDst: THPrint);
        INTEGER*4 PRJOBMERGE, xPRJOBMERGE
        PARAMETER (PRJOBMERGE=Z'8FD12400',
xPRJOBMERGE=Z'5804089C')

  * FUNCTION PrOpenDoc (hPrint: THPrint; pPrPort: TPPrPort;
  *                 pIOBuf: Ptr) : TpPrPort ;
        INTEGER*4 PROPENDOC, xPROPENDOC
        PARAMETER (PROPENDOC=Z'8FD92480',
xPROPENDOC=Z'04000C00')

  * PROCEDURE PrCloseDoc (pPrPort: TPPrPort) ;
        INTEGER*4 PRCLOSEDOC, xPRCLOSEDOC
        PARAMETER (PRCLOSEDOC=Z'8FD12000',
xPRCLOSEDOC=Z'08000484')

  * PROCEDURE PrOpenPage (pPrPort: TPPrPort; pPageFrame:
TPRect);
        INTEGER PROPENPAGE, xPROPENPAGE
        PARAMETER (PROPENPAGE=Z'8FD12400',
xPROPENPAGE=Z'10000808')

  * PROCEDURE PrClosePage (pPrPort: TPPrPort);
        INTEGER*4 PRCLOSEPAGE, xPRCLOSEPAGE
        PARAMETER (PRCLOSEPAGE=Z'8FD12000',
xPRCLOSEPAGE=Z'1800040C')


  * Note: prStatus should actually be passed as a pointer to
a TPrStatus
  *     record.
  * PROCEDURE PrPicFile (hPrint: THPrint; pPrPort: TPPrPort;
pIOBuf: Ptr;
  *                 pDevBuf: Ptr; VAR prStatus: TPrStatus);
        INTEGER*4 PRPICFILE, xPRPICFILE
        PARAMETER (PRPICFILE=Z'8FD124B2',
xPRPICFILE=Z'60051480')

  * FUNCTION PrError : INTEGER;
        INTEGER*4 PRERROR, xPRERROR
        PARAMETER (PRERROR=Z'8FD50000',
xPRERROR=Z'BA000000')

  * PROCEDURE PrSetError(iErr : INTEGER);
        INTEGER*4 PRSETERROR, xPRSETERROR
        PARAMETER (PRSETERROR=Z'8FD0A000',
xPRSETERROR=Z'C0000200')

  * PROCEDURE PrDrvrOpen;
        INTEGER*4 PRDRVROPEN, xPRDRVROPEN
        PARAMETER (PRDRVROPEN=Z'8FD10000',
xPRDRVROPEN=Z'80000000')

  * PROCEDURE PrDrvrClose;
        INTEGER*4 PRDRVRCLOSE, xPRDRVRCLOSE
        PARAMETER (PRDRVRCLOSE=Z'8FD10000',
xPRDRVRCLOSE=Z'88000000')

  * PROCEDURE PrCtlCall(iWhichCtl: Integer; lParam1,
lparam2, lparam3: LongInt);
        INTEGER*4 PRCTLCALL, xPRCTLCALL
        PARAMETER (PRCTLCALL=Z'8FD0A490',
xPRCTLCALL=Z'A0000E00')
```

```
  * FUNCTION PrDrvrDCE: Handle;
        INTEGER*4 PRDRVRDCE, xPRDRVRDCE
        PARAMETER (PRDRVRDCE=Z'8FD90000',
xPRDRVRDCE=Z'94000000')

  * FUNCTION PrDrvrVers: Integer;
        INTEGER*4 PRDRVRVERS, xPRDRVRVERS
        PARAMETER (PRDRVRVERS=Z'8FD50000',
xPRDRVRVERS=Z'9A000000')

  * PROCEDURE PrGeneral(pData: Ptr);
        INTEGER*4 PRGENERAL, xPRGENERAL
        PARAMETER (PRGENERAL=Z'8FD12000',
xPRGENERAL=Z'70070480')
  * PrGeneral is the gateway to the calls documented in
Inside Mac V
  * concerning added capabilities for printer resolution,
etc.
  * See the file PrGenDefs.inc and the demo program for
usage

  * PROCEDURE PrPurge;
        INTEGER*4 PRPURGE, xPRPURGE
        PARAMETER (PRPURGE=Z'8FD10000',
xPRPURGE=Z'A8000000')

  * PROCEDURE PrNoPurge;
        INTEGER*4 PRNOPURGE, xPRNOPURGE
        PARAMETER (PRNOPURGE=Z'8FD10000',
xPRNOPURGE=Z'B0000000')

  * FUNCTION PrStlInit (hPrint: THPrint): TPPrDlg;
        INTEGER*4 PRSTLINIT, xPRSTLINIT
        PARAMETER (PRSTLINIT=Z'8FD92000',
xPRSTLINIT=Z'3C04040C')

  * FUNCTION PrJobInit (hPrint: THPrint): TPPrDlg;
        INTEGER*4 PRJOBINIT, xPRJOBINIT
        PARAMETER (PRJOBINIT=Z'8FD92000',
xPRJOBINIT=Z'44040410')

  * FUNCTION PrDlgMain(hPrint: THPrint; pDlgInit: ProcPtr):
Boolean;
        INTEGER*4 PRDLGMAIN, xPRDLGMAIN
        PARAMETER (PRDLGMAIN=Z'8FDD2400',
xPRDLGMAIN=Z'4A040894')

  * end of prtrap.inc listing

  Listing Three: PrGenDefs.inc
  * File: PrGenDefs.inc   Definitions for calls to printer
trap PrGeneral
  *  1/11/88  Jay Lieske
  *
  * Information for GetRsl
  *     This file contains definitions for calls to PrGen-
eral with iOpCode=4
  *     (called gOpCode here) which is the GetRslData call.
  *

  * the GetRslData record is of length 128 bytes

        integer*1   GetRslDataRec(128)
        integer*2   gOpCode
        integer*2   gError
        integer*4   gReserved
        integer*2   iRgType
        integer*2   XRslRg(2)
        integer*2   YRslRg(2)
        integer*2   iRslRecCnt
        integer*2   rgRslRec(2,27)
                              ! defined as ARRAY[1..27] of
TRslRec
        equivalence (GetRslDataRec(1), gOpCode)
        equivalence (GetRslDataRec(3), gError)
        equivalence (GetRslDataRec(5), gReserved)
        equivalence (GetRslDataRec(9), iRgType)
        equivalence (GetRslDataRec(11), XRslRg)
        equivalence (GetRslDataRec(15), YRslRg)
        equivalence (GetRslDataRec(19), iRslRecCnt)
        equivalence (GetRslDataRec(21), rgRslRec)

        integer*4 pGetRslData
  C     parameter (gOpCode = 4)

  * You must set the value of gOpCode
  * in program. Get pointer to GetRslDataRec via:
pGetRslData = TOOLBX(PTR,GetRslDataRec )
  * and then CALL toolbx(PrGeneral, pGetRslData, xPrGeneral)

  * end of data for GetRsl

  * Information for SetRsl
  *     This file contains definitions for calls to PrGen-
eral with iOpCode=5
  *     (called pOpCode here) which is the SetRslData call.
  *

  * the SetRslData record is of length 16 bytes

        integer*1   SetRslRec(16)
        integer*2   pOpCode
        integer*2   pError
        integer*4   pReserved
        integer*4   hPrint
        integer*2   iXRsl
        integer*2   iYRsl

        equivalence (SetRslRec(1), pOpCode)
        equivalence (SetRslRec(3), pError)
        equivalence (SetRslRec(5), pReserved)
        equivalence (SetRslRec(9), hPrint)
        equivalence (SetRslRec(13), iXRsl)
        equivalence (SetRslRec(15), iYRsl)

        integer*4 pSetRsl
  C     parameter (pOpCode = 5)

  *  You must set the values of pOpCode = 5 and hPrint =
handle to Print Record
  * in program. Get pointer to SetRslRec via: pSetRsl =
TOOLBX(PTR,SetRslRec)
  * and then CALL toolbx(PrGeneral, pSetRsl, xPrGeneral)
  * end of data forSetRsl
  * end of file PrGenDefs.inc
```

Fortran's World

The Status of Fortran

Chuck Bouldin
Washington, DC
MacTutor Vol. 4 No. 11

Status of Fortran on the Macintosh

Until recently there has been a noticeable shortage of Fortran compilers for the Macintosh (translation: there has only been one Fortran compiler). This situation has changed enough that it is worthwhile to look over the available compilers and see what each has to offer. With the advent of the Mac II and Apple's new emphasis on "Desktop Engineering" there is now enough interest and cpu horsepower to transform the Mac into a serious "platform" for scientific and engineering applications. Finally, the Mac II is being treated by Apple and others as a serious low-cost alternative to more expensive workstations, so the availability of industrial strength Fortran compilers is an obvious requirement.

What's Available

At this writing, there are three Macintosh Fortran compiler vendors; Absoft, with a 68000 Fortran for the Mac+ and SE and a 68020 version for the Mac II and Mac's with accelerator cards. DCM has a Fortran with both 68000 and 68020 code generation, so it is suitable for both Mac+ and SE, as well as the Mac II. A third vendor, Language Systems, has just released a Fortran compiler that runs under MPW. Absoft is also in beta-test with a new Fortran (Risc Architecture Technology, or "RAT") that runs under Apple Unix and with MPW. The RAT compiler augments, but does not replace the existing MacFortran compiler, as it **requires** a 68020 and 68881. All of this products include inline hardware floating point code generation, a crucial factor for fast floating point calculations.

A Variety of Approaches

The compilers available (or soon available) for the Mac show a variety of different approaches to providing an "environment" for Fortran on the Mac. Each approach carries with it some inherent strengths and weaknesses. I want to go over these carefully before getting into comparisons of code size, execution speed, etc, since it is these differences in approach, **not** things like compile speed or code size that are frequently the limitation in the overall productivity of a programmer. There are inevitably subjective decisions in comparing these approaches, and I will make some. However, I will also try to be clear about **why** I make those decisions so that you can decide which of these products is most suited to your needs.

DCM Fortran

This is the only true "integrated" environment in the bunch. With this product you have an environment in which you can edit, compile, link, symbolically debug and run programs. The editor is about on a par with the MDS "Edit", but isn't nearly as high powered as, for example, Qued/M. The environment can be run under Multi-Finder, so if you want to use tools such as other editors or text utilities you can make them work reasonably well with this compiler. As with Absoft Fortran, this compiler allows both code and data to be larger than 32K in size. There are a number of extensions to ANSI Fortran, mostly improved flow control with SELECT CASE and other constructs.

The classic dilemma of an integrated environment is that if you don't like the setup as the vendor provides it, then it is usually difficult to modify. I find this environment marginally acceptable, as the editor seems under-powered compared to stand-alone alternatives and there are also minor problems with the Mac interface. For example, during compiles the cursor is not consistently maintained with the watch icon, so the user has no visual feedback that anything is happening. Worse, the edit windows do not have zoom boxes. Finally, this compiler has the ultimate "dumb" linker. A link simply appends the 155K runtime package to your compiled code. So the minimum size for any program is about 156K! You don't have to link in the runtime library, as it can be auto-linked at execution, but the size of this runtime package will be a real detriment to anyone seeking to write applications to distribute.

Absoft MacFortran

This compiler is now in beta-test in the 2.4 version. This is much improved over the 2.3 release, with a number of bug fixes and a much better Mac interface. The compiler can now be configured so that your choice of text editor, as well as the Fortran linker, Debugger, Scriptbuilder, etc, can be "sub-launched" from the compiler. This does a lot to speed up transfers between the pieces of the programming environment, and also allows the programmer flexibility in choosing those pieces, such as specifying your favorite text editor. A script-builder utility similar to the MPW Commando is now provided to tame the atrocious command line interface which was the only way to run the Linker in the 2.3 version. In addition, Scriptbuilder can be used to construct a list of files for doing multiple compilations without operator intervention. The runtime library and user pre-compiled subroutines can be automatically linked at execution, so during development it is almost never necessary to use the linker. As with DCM Fortran, linking the runtime library to your code just copies in the entire library, but in this case you only have to carry around 12K for the runtime package. There are extensions to ANSI Fortran that are similar to those in DCM Fortran. Also in common with DCM, Absoft Fortran has no restrictions on code or data sizes, other than available memory.

This compiler has developed a bad reputation for bugginess. The 2.4 beta release fixes a number of these, and Absoft has taken

© The Definitive MacTutor, Vol. 4

steps to improve feedback from users by establishing a dial-in bulletin board that users can access to obtain information, fixes and to report bugs. Along with the clean up of bugs and improvement in the Mac interface in the 2.4 version, this compiler appears to be much improved. The compiler does not correctly support the Fortran ENTRY statement, but as far as I know, this is the only deviation from the ANSI standard for Fortran.

Language Systems Fortran

This is a Fortran that runs under MPW. This brings with it all the power and tools of MPW, such as cross-linking with C and Pascal, support for MacApp and (Someday! Are you listening, Apple?) the common symbolic debugger that will work with all these tools. This is definitely a power environment for serious Fortran development work, but it also implies that you carry around the baggage of MPW whether you want it or not. In practice, you actually need to learn very little about MPW to get started. Once script files to compile and link your programs are developed, the MPW environment for Fortran is reasonably "Mac-like". There is a Commando interface for the compiler, so it is easy to get started. Language systems also provides calls to Mac Toolbox routines as inline constructs rather than through a "pointer" subroutine, as in DCM and Absoft Fortran. Finally, this is the only compiler that supports Structures and Records, so that complicated data constructs are handled by the compiler, rather than arduously mapped onto arrays by the programmer. This is an important consideration for accessing the Mac Toolbox since this always requires data structures, some of which are complex. This compiler supports 68020 and 68881 code generation. Because of Apple's decisions the MPW compiler is stuck with the 32K segment limit on code, although they allow data to exceed 32K in size.

Absoft RAT Compiler

This is a new Fortran that runs under MPW. The interesting thing about this compiler, which is still in early beta testing, is that the compiler does some optimizations that are usually associated with RISC (reduced instruction set chip) microprocessors. The compiler requires both a 68020 and 68881. The compiler treats the register pool of both processors as one set of registers, and the contents of these registers are allocated dynamically with a score-boarding technique. What this means is best illustrated by an example. A compiler will ordinarily choose, automatically or under programmer control (as with the Register keyword in C), certain variables that are kept in cpu registers. This optimization is done on a module-wide basis. That is, registers allocations are constant over the execution of a subroutine. This is fine until you run out of registers. For example, if a compiler has three registers left to allocate when it compiles a code structure of three nested DO loops, then the loop counters may be put into registers. However, if the compiler now encounters another set of triply-nested DO loops farther along in the same subroutine, it is usually impossible for those counter variables to go into registers. The departure that the RAT compiler takes is that registers contents are optimized on a line-by-line rather than subroutine-wide basis. In the above example, the first three loop counter variables could go into registers and then be flushed out and replaced with the next set of three loop counters. This register score-boarding method should help most in code that is large and complicated, with lots of variables, which where a traditional compiler will run out of registers variables to statically allocate.

As in Language Systems Fortran there is a Commando interface for the compiler and calls to Mac Toolbox routines are inline calls. However, there are no Structure or Record data types, so one must still use array declarations and equivalence statements to pass arguments to Toolbox routines. Also, the RAT compiler will not be available until Apple releases MPW 3.0, as Absoft is waiting for Apple to provide a fix for the 32K data segment limit, rather than coding around Apple's mistake as Language Systems was forced into.

Benchmarks

With Fortran, there are a number of quasi-standard benchmarks that have been run on almost all machines. These are not Mac specific, so they do not test the ability of the compilers to provide support for the Macintosh environment, but they do have the advantage that they have been run on many other machines, allowing broader comparisons to be made. I chose to evaluate these compilers with the three most common tests: the Sieve of Eratosthenes, the "Whetstone", and the double-precision Linpack. These tests emphasize, respectively, integer operations and addressing of small arrays (Sieve), floating point operations and transcendental function evaluation (Whetstone) and solution of linear equations, double-precision floating point arithmetic and addressing of large arrays (Linpack). In addition I kept track of compile times and code sizes of these three programs.

All compilers were run with options set to maximize compile and execution speed. All compilations were done with INTEGER*4 as the default storage class, and I took no pains to tweak options to change code size. All tests were run on a Mac+ with a Novy systems 16 mhz 68020/68881 upgrade board with 4 megs of 1 wait-state memory, a 512K disc cache and a MacBottom 45 hard disc with 26 msec seek time. In computation speed, this system is roughly the same speed as a Mac II. The results are tabulated below. Each position in the table contains, from top to bottom, the results for DCM Fortran 3.0 , Absoft MacFortran 2.4 , Language Systems Fortran 1.1, and Absoft RAT Fortran "pre-beta". All sizes are in kilo-bytes and all execution times, except Whetstone, are in seconds. Whetstone results are traditionally reported in units of "whetstones per second". For this benchmark only, a larger number means faster execution. Finally, for Whetstone and Linpack, I included speeds for a VAX 11/780, so you can see how well this desktop computer is doing against a large mainframe!

Results of Generic Fortran Tests

These test results need to be interpreted with some care. For example, an important distinction in the Compile and Link time results is that DCM Fortran and MacFortran do an automatic link-at-runtime, which is the time reported in the table, while RAT and Lang. Sys. both use the MPW linker, so they always require an explicit hard link. This is one example of the compro-

mises that one makes by running under MPW rather than your own integrated environment.

On compile speed, it is a race between DCM and MacFortran. DCM is the fastest turnaround for very small code, but MacFortran starts to edge it out as code size grows. The RAT compiler is competitive on compile speed, but both it and Lang. Sys. suffer from link times under MPW.

The Absoft RAT compiler sweeps the field on execution speed, a reflection of the superior optimization features of this compiler. As predicted, the register scoreboarding technique has little effect on Sieve, where there are so few variables that static register allocation is virtually as good as dynamic optimization. On Linpack and Whetstone, the optimization in RAT makes a marked difference. Averaged over the three tests the other compilers are comparable in execution speed.

Code size is a complicated issue, as this is where differences in "philosophy" will show up most glaringly. MacFortran and RAT are the winners for small code size, partly because of the small runtime libraries. This is a minimalist approach, trading off small code size for little built in Mac support. For example, in MacFortran the runtime library only supports a non-scrollable, non-resizable "glass TTY" window and no support for desk accessories or other parts of the Mac environment. Language Systems and DCM provide more built in Mac features, which accounts for some of the code size. The Language Systems runtime support includes a scrollable text edit window with the usual File, Edit and Apple menus. If you want a reasonable built-in Mac interface, this looks to be a reasonable compromise.

Mac Programming Support

The test that I ran are generic Fortran code, so how do we evaluate the Fortran compilers on their ability to generate real Mac-style applications? Generally speaking, it is very difficult, and often inappropriate, to attempt a full-blown Mac interface in Fortran. The main difficulty is that all the toolbox routines expect Pascal or C type data structures, which are not supported in most Fortran compilers and are not part of the ANSI 77 standard for Fortran (although they are in the proposed 88 standard). Tools for adding a Mac interface are therefore an important consideration in choosing a Fortran compiler.

Fortran programmers have two main choices: (1) Use a Fortran that runs under MPW and program the interface code in C or Pascal, or with MacApp and link to your Fortran code, or (2) Use a programming aid called FaceIt. FaceIt is a set of stand-alone code resources that can be easily added to your Fortran code and makes it trivial to add Macintosh look and feel to your programs. FaceIt is presently available for MacFortran, and Language Systems Fortran, and should be available with RAT.

Conclusions

But first, a disclaimer. I have used the Absoft MacFortran for over 3 years, so I am very familiar with that product. In contrast, I have used an evaluation copy of DCM Fortran for a few weeks, have used the Language systems Fortran for about a month, and used RAT just enough to get the 3 benchmark programs run. I have tried to be objective, but it is almost inevitable for two things to happen: (1) I may tend to prefer MacFortran simply through familiarity. (2) I am much more familiar with bugs and flaws in MacFortran because of more accumulated usage with it.

With that settled, these are my conclusions. On a "by-the-numbers" basis, MacFortran and RAT are clear winners in the categories of stand-alone and MPW compilers. However, there is a lot more to overall productivity than the numbers in the table. Since I was trying to measure raw performance, the tests that I used are all "vanilla" Fortran and do not include any Mac features at all. In terms of runtime support of the Mac, MacFortran runs last, behind DCM and Language systems. If you want any Mac features, such as scrolling windows, DA support, etc, in MacFortran you do it all yourself. DCM and Language Systems are putting more code into the runtime package (hence the larger code sizes) to support the Mac environment. For applications that have a Fortran written Mac interface, the code sizes would probably be closer than the examples that I chose. However, as I indicated above, my approach is to avoid doing a Mac interface program in Fortran, and, instead, to use other tools which are better suited for that purpose.

The choice of stand-alone compiler, integrated environment or MPW also has important consequences. For instance, I found all of the compilers to be very responsive in terms of compile speed and context switching between programming tools. But, I am running on virtually the fastest Mac hardware available. On a floppy-based Mac+, the integrated environment of DCM

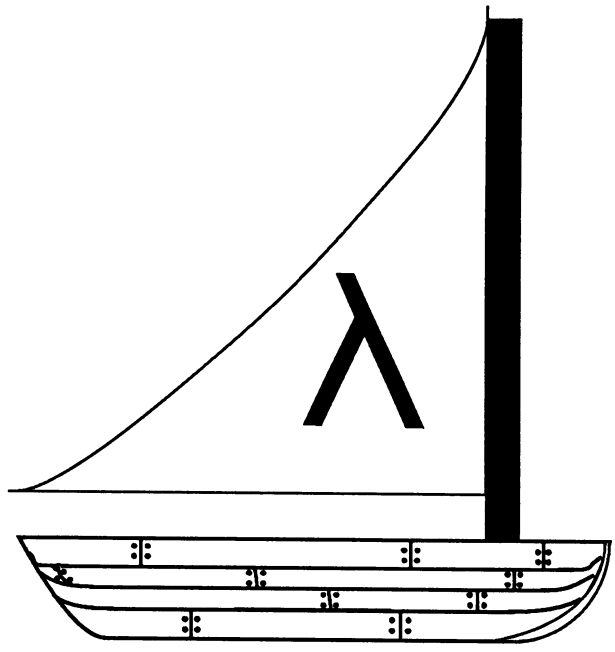
Sieve	Compiler and Link	Linked Code Size	Execution Time
Sieve	5.2 sec (DCM)	155 Kbyte	12.1 secs
	9.5 sec (MacFortran)	13 Kbyte	9.8 secs
	29.0sec (Lang. Sys.)	53 Kbyte	13.5 secs
	8.6 sec (RAT)	20 Kbyte	9.1 secs
Whet.	11.5 sec (DCM)	160 Kbyte	632K whets
	11.6 sec (MacFortran)	16 Kbyte	545K whets
	40.2 sec (Lang. Sys.)	55 Kbyte	540K whets
	23.0 sec (RAT)	22 Kbyte	830K whets
		1054K whets (VAX)	
Linpack	14.5 sec (DCM)	171 Kbyte	16.0 secs
	13.2secs (MacFortran)	19 Kbyte	12.4 secs
	52.8 sec (Lang. Sys.)	59 Kbyte	12.4 secs
	26.0 sec (RAT)	24 Kbyte	9.2 secs
		4.9 secs (VAX)	

Figure 1. Test Results

could be much more important. In the other direction, running under MPW confers the benefits of use of Mac-App or cross-linking with C or Pascal. This is not an unmixed blessing, however, as it ties the fortunes of the compiler to decisions made by Apple. For example, the 32K segment limit for data has not yet been eliminated and there is still no symbolic debugger for compilers running under MPW.

The bottom line is that you must choose amongst these products on the basis of your needs. On systems that cannot support the overhead of MPW, or if just don't like MPW, you may well be happier with MacFortran or DCM. If you have an existing Mac interface in C or a commitment to MPW, then Language Systems offers a means to integrate mainframe Fortran code into a Mac interface shell, with lots of MPW tools to help you.

I suspect that Fortran on the Mac is now going to see some of the improvements that we saw take place with C compilers once a competitive marketplace was established. We may even see programmers using a synergistic approach with several different Fortran compilers, just as some developers presently use Lightspeed C and MPW C at different times for different purposes.



Lisp Listener

Lisp Listener

MacScheme vs Coral Lisp

MacScheme

Paul Snively
ICOM Simulations
MacTutor Contributing Editor
MacTutor Vol. 4 No. 3

Paul Snively is one of our favorite people; a long time MacTutor fan, and contributor, we are glad to have this contribution from him. Here are some notes Paul sent that will bring all his fans up to date on what he has been doing.

Letter to MacTutor

"Well, here it is—after a period of time that was entirely too long on two counts (too long both in terms of time between payment and delivery (roughly nine months) and time between receipt of software to support this article and delivery (about four months)). Good grief; look at those parenthesis—you can tell I've been working in Lisp lately!

Let me congratulate you on the apparent ongoing success of MacTutor as an organization, and upon the consistently high overall quality of the content (my own work notwithstanding). The article on the writing of custom printer drivers was particularly welcome, and there is much more than could be done there. In particular, a LaserWriter driver that doesn't require the "Laser Prep" file (i.e. one that generates pure PostScript without any Apple-proprietary PostScript macros) would be a most welcome project from a number of standpoints, I think.

In case you might be curious as to what I've been up to lately, ICOM Simulations, Inc. has kept me pretty busy from day one, although the situation seems to be intensifying lately. Early on in my time with ICOM, there was "TMON Plus/TMON Professional" (depending upon whom you talked to) to work on. Unfortunately that project met an early demise when it became clear that Waldemar would not be able to complete the kernel before he had to return to MIT after the summer.

After that we concentrated on TMON 2.8, which was released at Boston. Unfortunately, it isn't everything that we might wish that it was (in particular, the lack of disassembly of '020 and '881 opcodes is hurting us, as is the funny-looking display when you're on a Mac II in multiple bit-planes). However, our customers seem happy to have something that at least will run on a machine with an '020, and it does have a few additional nicities (such as the INIT loader, the Programmer's Key INIT, better documentation, no VBL tasks while in TMON, etc.)

Lately, and in some ways concurrently with all of this, we continue to write MacVentures. There's a certain irony in being the developers of the *de facto* standard Macintosh debugger and also of a series of adventure games. Apple makes sure that we're kept on top of new System Software developments (we tested Juggler, of course, for some time) because of TMON, but the applications that benefit from the knowledge first are a bunch of games! (I'll bet we have the only MultiFinder-aware adventures on the market.) Also ironic is the fact that the MacVenture kernel seems to be a good testing environment for all kinds of weird issues like window management, application-defined events, synchronous and asynchronous I/O, and the like.

Speaking of which, here's a bug note for MultiFinder 1.0 that you may want to publish. If your application is MultiFinder-aware and has the canBackground bit set in its SIZE resource (i.e. it will run in the background under MultiFinder), and the application posts an application-defined event (appEvt1-appEvt3; appEvt4 is for Suspend, Resume, and mouseMoved events), the event will not necessarily get sent to the application that created it—it may go to some other application running under MultiFinder. Needless to say, this can cause your application and the other application all kinds of headaches if you rely on application-defined events to work properly (the MacVentures historically have used appEvt1 and appEvt2 for won/lost events and sound events; the kernel has been rewritten to be a state machine now that uses no application-defined events).

It's really been fun Lisping lately; I'd forgotten how nice it is to work in a good Lisp development environment. It's been so nice and so accessible that I've gotten some really interesting ideas that I think (and in some cases know) could be implemented fairly easily in Lisp. One thing that comes immediately to mind is a Display PostScript implementation (Lisp is great for writing interpreters). Another thing that comes to mind is porting Xerox's "NoteCards" program from their Lisp workstations to a Mac II running Common Lisp. The world seems to love HyperCard, so they should faint over "NoteCards."

Well, again, thanks for everything! Please say hello to the rest of the gang for me (and please tell Shelley that I enjoyed talking to her again very much). Hopefully I'll see you in August!" *Sincerely, Paul Snively*

The Cheap AI Workstation

The Macintosh II has been around for a few months now, and much of the furor has died down, aside from gripes from disgruntled Macintosh II owners that Apple has been less than speedy in coming out with their own color monitor, or that more SIMMs are hard to come by. In some ways, the honeymoon is over, and we're beginning to hear the complaints. People aren't as blown away by the performance of the 68020/68881 as they thought they would be. Some software, particularly those packages that insisted on breaking some of Apple's clearly-defined rules, doesn't work on the Macintosh II. Virtually all games that do animation of any sophistication at all don't work because they use the alternate screen buffer, which the Macintosh II hardware lacks.

© The Definitive MacTutor, Vol. 4

My personal opinion is that, at least for developers if not for the end users, the honeymoon for the Macintosh II is ending before the marriage has even been consummated. I feel that there's been too much emphasis on why the Macintosh II is perhaps not as great a "souped-up Mac Plus" as some people might have wished, and not enough emphasis on why the Macintosh II is probably the most bang for the buck that you will be able to get in a low-end workstation for the next few years.

When I was a computer science student in college, I thought the academic scene was pretty boring. By the time I was a freshman in college I'd been programming for four years already. Taking courses called "Advanced Programming Techniques" that consisted of learning the PL/I programming language and learning about such ostensibly innovative data structures as "linked lists" and the algorithms to manipulate them impressed me about as much as learning Latin would have as a high-school student (in fact, I find Latin a great deal more applicable to real-world situations than PL/I)!

That was the obvious side of my college experience. The non-obvious side of my college experience consisted of getting to know two fascinating individuals and the language in which they seemed best able to express their concepts. The people were Douglas Hofstadter, author of *Gödel, Escher, Bach: An Eternal Golden Braid*, and Daniel Friedman, author of *The Little LISPer*. The language in question was, of course, Lisp, or at least one or two interesting variations on it.

I learned Franz Lisp and Scheme on the VAXCluster at school on my own time. It was a wonderful environment to work in, and I spent many hours in front of my Model I TRS-80, which was connected to the phone in my dorm, and from there connected to the computing network on campus. I had access to a wonderful Lisp structure editor, and, Lisp being the kind of language that it is, I spent a fair amount of time customizing my copy of the editor to suit my tastes. It was fairly easy, if also fairly dangerous, because I could literally change the editor, and as soon as I entered the change, it would take effect. Needless to say, it was quite possible to goof and do something to make the editor totally unusable this way, so working on a backup copy was always a good idea. But to me, Lisp was always the ultimate interactive development environment, and the concepts behind it were extremely interesting.

By now you're probably wondering where all of this is leading up to. Well, bear in mind that I was working on a large VAXCluster with lots of disk space and lots of memory—and lots of horsepower. I got a Lisp for my TRS-80 at one point, but it turned out to be a very frustrating thing, what with the limited RAM and CPU power of most 8-bit micros. I basically resigned myself to never working in such an environment again, and since I had become entrenched in the microcomputer world, I figured that I'd never have a good Lisp system.

Time went on, and although decent Lisp systems got smaller and created a whole new class of machines called "workstations," which had names like Symbolics and Lisp Machine, those, too, were machines with a lot of horsepower and RAM and disk space, dedicated to doing one thing: running Lisp, and running it well enough so that professional AI researchers could do meaningful work. They were obviously still way out of my league as an interested amateur, with price tags well into the five figure range.

But the Macintosh II exists now, a machine with an MC68020 microprocessor, an MC68881 math coprocessor, anywhere from 1 to 8 megabytes of RAM, and as much disk space as you can afford. It's not a dedicated Lisp machine, but with a couple of Macintosh products available, it might as well be, and it's a lot cheaper.

I picked up two Lisp implementations at the Boston MacWorld Exposition. One of them is a newcomer, released at the show. The other has been around somewhat longer, but has undergone some evolution and deserves a serious look. What is perhaps most interesting about the two products is that while they both cover extremely similar ground, because of their underlying philosophies they will appeal to different people and for different reasons.

MacScheme+Toolsmith 1.0 is the older of the two products that I have, or at least MacScheme is. In chapter one of the documentation, it says: "MacScheme, introduced in August 1985, was the first microcomputer implementation of any modern Lisp standard." It's a rather bold statement, and one open to question if you have used some of the other Lisp offerings on other machines, particularly any of the Golden systems on the IBM PC-class computers. On the other hand, if you accept Semantic Microsystems' statement that "Scheme is one of the two major dialects of Lisp," and understand that the other one is Common Lisp, and also know that until very recently there was no such thing as a complete microcomputer implementation of Common Lisp, then the statement is certainly true.

MacScheme historically has been a semi-compiled, byte-code oriented system, but with the addition of the Toolsmith code, MacScheme has become a language capable of compiling to native code and creating standalone applications written entirely in Scheme. Scheme aficionados should find this very exciting. Complete source code for a simple text editor is included to help show how to create applications. The choice of a text editor is an interesting one; I guess the message is that "Lisp isn't just for AI anymore," which is certainly a valid point.

If there's a general way to describe Scheme, it has to include the word "elegant." Scheme was designed with an eye toward simplicity and power. It tends to be the smallest implementation of Lisp for any given machine, and MacScheme+Toolsmith is no exception to that rule; the kernel is about 75K in size, and a goodly-sized heap image is about 182K. MacScheme+Toolsmith fits pretty well in a one megabyte machine, as long as you're careful about hogging up valuable RAM with things like debuggers and INITs and sounds and so forth.

MacScheme+Toolsmith 1.0 is shipped on three 800K disks, labelled "Toolsmith," "Library," and "Miscellaneous." Don't be misled by the fact that the whole product takes 2,400K of disk space to ship; you can copy the Toolsmith disk, boot from it, and accomplish a great deal without ever looking at the other two

disks. The Toolsmith disk contains a System Folder, the MacScheme+Toolsmith application, a generic Scheme heap image file, and a Toolsmith heap image file. Double clicking the application will cause MacScheme+Toolsmith to look for "Scheme.heap;" you can double-click any heap image file to cause MacScheme+Toolsmith to load it instead.

Once MacScheme has come up, you'll see one thing that immediately sets it apart from Allegro CL, the other Lisp environment that I wish to look at: MacScheme has a fairly spartan user interface. It's too easy to be put off by the minimalist appearance of the product. Avoid the temptation. While the tools to tinker with MacScheme may not be as obviously up front and "Macish" as they are in Allegro CL, they are, for the most part, there. MacScheme is a complete Scheme implementation, and, like Scheme as a dialect, is minimalist in philosophy, preferring to be extended via available source code (of which there is *plenty* with MacScheme+Toolsmith) rather than providing everything up to and including the kitchen sink as part of the language kernel, which is more Common Lisp's style.

MacScheme offers a transcript window and four menus as its user interface upon booting. The transcript accepts invocations of Scheme expressions and returns their value in a fashion that should be familiar to anyone who has used a command-line interface before. The four menus are pretty much what you'd expect, especially considering that any Macintosh application that claims to follow the user interface guidelines must have the first three. The four are the Apple, File, Edit, and Command menus.

The Apple menu simply has the standard fare: an "About..." box that tells a bit about who's behind MacScheme with some copyright information, and whatever desk accessories are available on your system.

The other menus can, and do, vary somewhat from application to application, so let's look at them more closely. Here's the File menu:

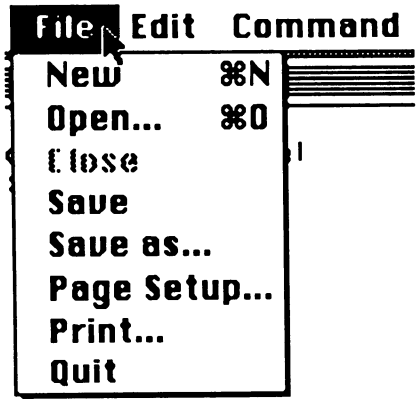


Figure 1

There's certainly nothing radically different about this File menu; it contains pretty much what you expect to see in a Mac application. "New" creates a new Scheme source file for you to put code in. "Open..." opens an existing source file. "Close" will close the current file if there is one. "Save" saves the file's contents. "Save as..." allows you to save a file's contents in another file. "Page Setup..." allows you to define your printing options. "Print..." prints the file's contents on the chosen printer, and "Quit" ends your Scheme session.

Here's the Edit menu:

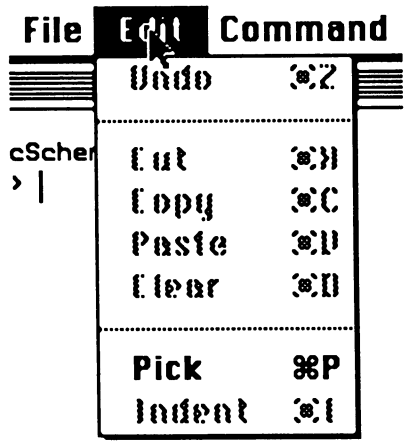


Figure 2

All of the Edit menu's functions are standard with the exception of the last two: "Pick" and "Indent." Both of these functions are Scheme-specific. "Pick" uses a simple heuristic to pick the nearest Scheme expression. For example, if the vertical bar is blinking behind a closing parenthesis, using "Pick" will select back to the matching open parenthesis, assuming that the parenthesis enclose a complete Scheme expression. Note that there are a couple of ways to invoke Pick: you can use the menu or the Command-key equivalent.

"Indent" can be used when you're editing some Scheme code to reformat its appearance. Like most Lisp editing systems, the editor in MacScheme will automatically indent as you enter the code so as to keep it readable. This is fine for when you're entering the code from scratch, but editing can ruin the format. Hence the necessity for something like "Indent," which operates on the current selection. The Tab key also serves this purpose.

Now let's look at the Command menu:

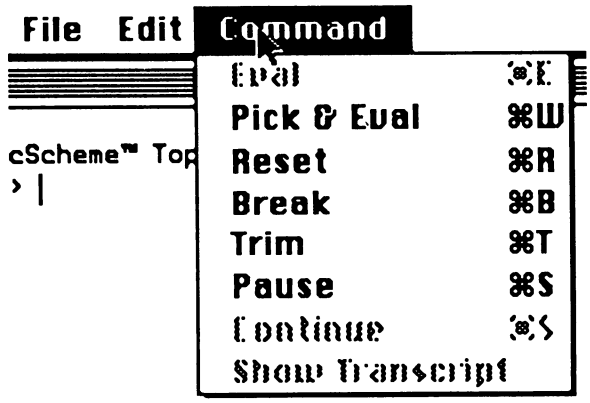


Figure 3

This menu is all Scheme specific, as you might expect. The first function is the most obvious one for a Lisp system: "Eval."

© The Definitive MacTutor, Vol. 4

It *eval*uates the currently selected Scheme expression, printing the value in the transcript.

The second item, "Pick & Eval," is the most frequently used one. It does what it says, combining the functionality of "Pick" from the Edit menu and "Eval" just mentioned. Since this combination is so frequently used, it can be invoked three ways: by choosing the menu item, by pressing the Command-key equivalent, or by pressing the Enter (*not* Return) key.

The next item, "Reset," is handy when some process has been interrupted and seems irrevocable. Resetting the system is the next best thing to quitting and starting again. It eliminates all your definitions and side effects from the time you brought up MacScheme.

Next comes "Break." This is used to interrupt a Scheme process. It puts you into MacScheme's debugger so that you can examine variables, the stack, or results of evaluating expressions. You can continue execution from the debugger as well.

"Trim" is a housekeeping function that cuts off all but the last page of the transcript. It's useful for when the transcript gets too long and seriously degrades the performance of the system.

"Pause" allows you to stop whatever is going on so that you can see what's on the screen. "Continue" is its counterpart, and doesn't become active until "Continue" has been used. They have the same Command-key equivalent so that you can pause and go with a single Command-keystroke.

"Show Transcript" is a convenient function for bringing the transcript window to the front when you have several windows open. That's a desirable thing, since that's where all the action is.

Believe it or not, that's really all there is to MacScheme's user interface. Of course, with MacScheme+Toolsmith, you have access to the Toolbox and, in the case of some user interface elements such as menus and windows, you have a pleasant pseudo-object-oriented interface to them, so you can certainly add your own menus with your own tools; indeed, I'd recommend it. In particular, if someone has a Scheme version of Common Lisp's "Apropos," I'd certainly love to see it.

Now let's contrast this user interface with that of Allegro CL. Before I do, I'd like to point out to those who might be wondering why I'm concentrating so much on user interfaces that I am an environment freak, which is to say that I believe that the programming environment is what makes the system usable/more productive/more fun or what have you. A good example of a *great* environment to work in is Smalltalk-80's. Smalltalk-80 as a language is a very nice object-oriented one with a lot to recommend it—things like consistency, flexibility, and power. Smalltalk-80, however, is also very huge and very complex by most language standards. To make matters worse, programs that are written by a Smalltalk-80 user simply become part of the overall environment. There isn't any distinction made between code that the language provides and code that the user writes. So if you extend an already huge language by adding several megabytes of user code, you wind up with a potentially larger mess.

That's why Smalltalk-80 includes several tools for doing what, in total, I call "managing complexity." In the most general sense, anytime a programmer creates a tool or environment that doesn't have a direct effect on the functionality of the code that s/he is writing, s/he is managing complexity (MacApp, for example, is a very good attempt to manage the complexity of writing a Macintosh application with a good, reliable, standard user interface).

There are essentially two schools of thought on programming environments: the school that says "thou shalt conserve disk and memory space and gain development efficiency and productivity by disciplines such as structured programming," and the school that says "given sufficient disk and memory space, gain development efficiency and productivity by having all system source online at all times; have a programmable, customizable editor; have excellent stepping, tracing, stack frame analysis, and other debugging tools; have tools that make the many megabytes of mess easy to access so as to get what you need, and have all this in one consistent environment." (Of course, some people will argue that there are two kinds of people: those who believe that there are two kinds of people and those who don't, but that is another story.)

The first school of thought is that espoused by virtually every common development environment in use today. Think about it. Disk space is considered an expensive "plus," with memory being treated similarly. Most popular languages today (Pascal, C, assembler) do not encourage the idea of extension of the language by the user; languages that do (FORTH, Smalltalk-80, Lisp) are out on the "lunatic fringe" of programming society. Virtually no development environment includes its complete source code (and, to be fair, there are valid commercial reasons for wishing to avoid this), but Smalltalk-80, for example, likes to present you with complete source code online at all times—one reason that it usually takes at least 10 meg of disk space to run Smalltalk-80. High-level implementations for the Smalltalk-80 byte-code primitives (the things actually implemented in the machine language of the target computer) are even included so that you can see how the algorithms work.

When dealing with such large quantities of code, something more than a classical text editor is needed. Smalltalk-80's answer to this is the browser, which is a special kind of editor that divides all of the code in the system into manageable chunks (in Smalltalk-80's case, this is facilitated by its object-oriented nature). A simple means of getting to any particular piece of code in the system is provided.

Common Lisp is the Lisp world's answer to Smalltalk-80, and it tends to reflect that fact by being somewhat larger than other implementations. To give you some idea as to what I mean by this, the Allegro CL application just barely fits on an 800K disk. Contrast this 750+K kernel to MacScheme's 75K.

You may wonder why an implementation like Allegro CL is so large. Most of the answer to this question can be found by looking at a copy of *Common LISP: The Language*, by Guy L. Steele, Jr. This book completely defines the Common Lisp language standard, and it's not the sort of thing you're going to

finish reading over a, say, six hour airline flight unless you're an incredibly fast reader.

At least two or three other Lisps for the Macintosh (the public-domain XLISP, and ExperLisp and ExperCommonLisp, the latter two from ExperTelligence) have been subsets of Common Lisp to one extent or another, and some people who use a complete Common Lisp on a workstation, mini, or mainframe at work are leery of these products, particularly if they plan to convert their Common Lisp code to run on the Macintosh. Users of Allegro CL have fewer worries, since Allegro CL is the only complete implementation of Common Lisp on any mass-marketed microcomputer as of this writing. (When doing conversions, you still have to worry about the things you always have to worry about, even when going from Common Lisp to Common Lisp—things like macro translation, what packages are available in any given implementation, etc.)

So what does this Common Lisp environment look like? Let's find out. First, the menus:

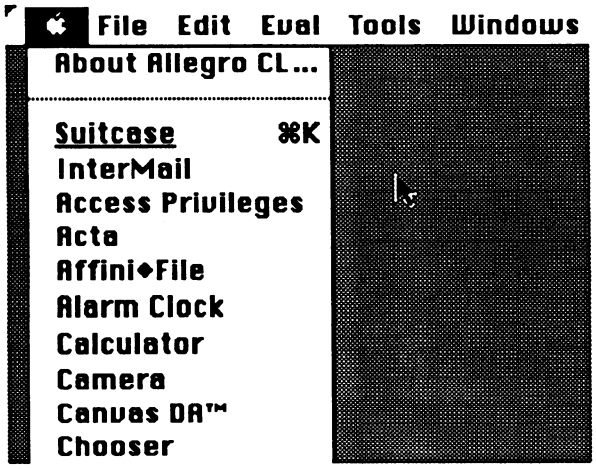
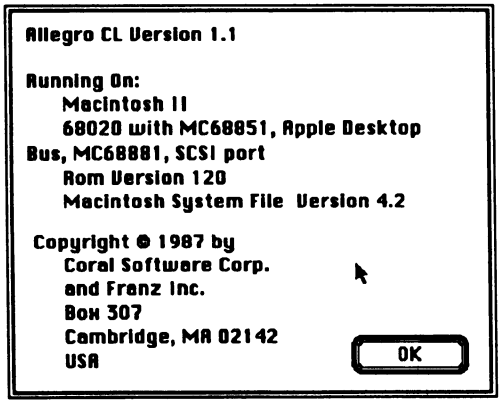


Figure 4

This is a pretty standard Apple menu, with the "About..." item for Allegro CL and, in my case, oodles and oodles of desk accessories, thanks to Prof. Mac (aka Steve Brecher) and Suitcase. The "About..." box for Allegro CL is a little unusual because it does more than just tell you about Allegro CL; it also tells you what you're running on:



Well, it isn't perfect. First of all, when it has too much information to present, it wraps it around without re-indenting, which isn't good aesthetically. More importantly, it thinks that my Mac II has an MC68851 installed, which it definitely does not.

Next is the ubiquitous File menu:

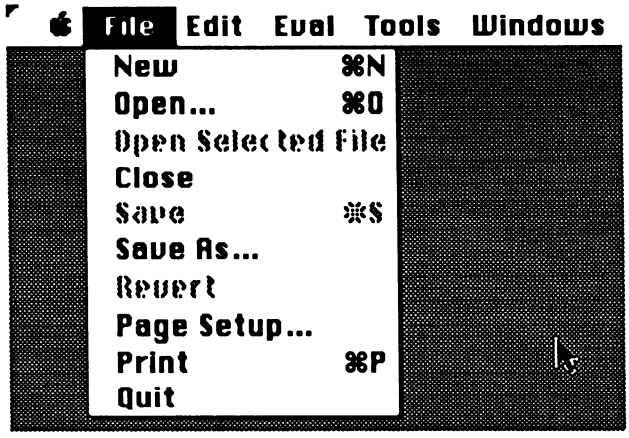


Figure 6

This looks a lot like the File menu from other development environments, with the typical "New," "Open...," "Close," "Save," "Save As...," "Page Setup...," "Print," and "Quit" choices. The "Open Selected File" and "Revert" choices are desirable but not found in most environments, so seeing them here is a pleasant surprise.

Next is Edit:

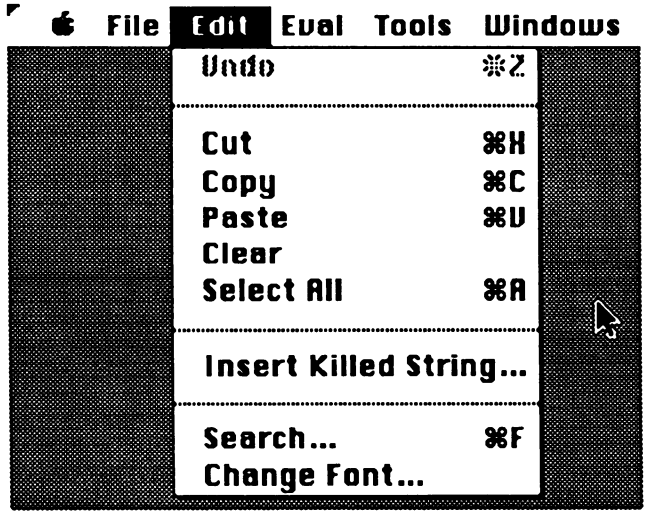


Figure 7

Here's another menu with few surprises, with the possible exception of "Insert Killed String..." which relates to the Allegro CL editor, called FRED (more about FRED later). I won't insult your intelligence by going through all of these common functions.

Now for Eval:

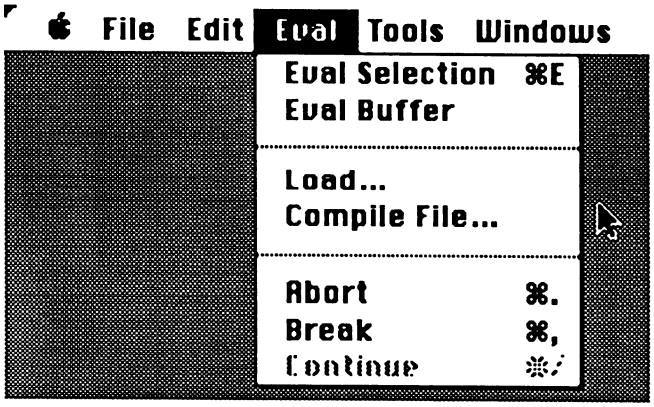


Figure 8

This menu is a little more Lisp-specific, but still pretty intuitively apparent. "Eval Selection" does just that, and it's not particularly picky about where the selection is, either. If the selection is a valid s-expression, it will be evaluated.

"Eval Buffer" is handy when you've made some fairly extensive changes to a source file. Just make the file's window the active window and choose this menu item. It's equivalent to choosing "Select All" from the Edit menu, then choosing "Eval Selection." This probably won't be used often, however, since Lisp is incrementally compiled, which encourages editing and recompiling individual definitions since the effects are system-wide, rather than forcing you to recompile all definitions following the one that you changed in the source code.

"Load..." puts up the Standard File Dialog showing Allegro CL source files so that you can load one. Loading a file is simply a quick way of opening it and eval'ing the buffer; the only difference is that it's a one-step operation and doesn't create a window for the file in question.

"Compile File..." allows you to create a file of Lisp that's compiled down to MC68000 machine code. Such files have the extension ".fasl" as opposed to ".lisp," and have the advantage of loading considerably faster than their textual counterparts, as well as making it tough to figure out how something does what it does (Coral Software distributes their Dialog Designer software with Allegro CL 1.1 as a ".fasl" file; apparently the Dialog Designer is proprietary). Note that Allegro CL does not currently create standalone applications, though Coral has promised that capability for the near future.

The last three items are familiar and basic debugging tools, good for stopping a Lisp process in midstream, tinkering around in the current lexical context, and continuing (or not) as you see fit.

"Tools" is where most Allegro CL users will probably find themselves spending some time:

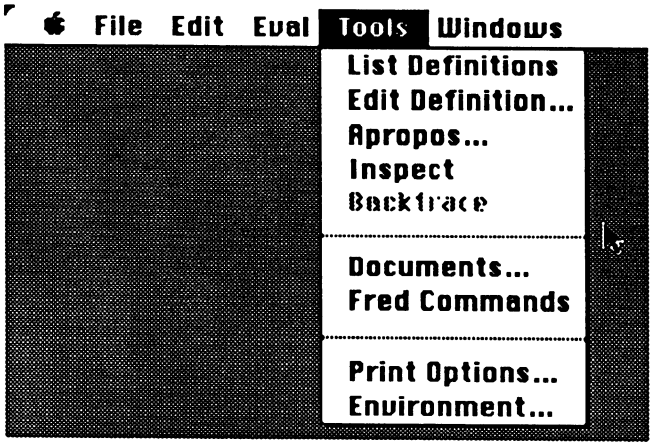


Figure 9

The first item is rather interesting; it's a function that lists all definitions within a particular FRED buffer (whichever one is active when the menu item is chosen). In order to do this, it must go through the entire buffer looking for DEFUN and DEFOBJFUN and so on (can you say "heuristic?" I knew you could)! It then creates a dialog box containing the names of the definitions:

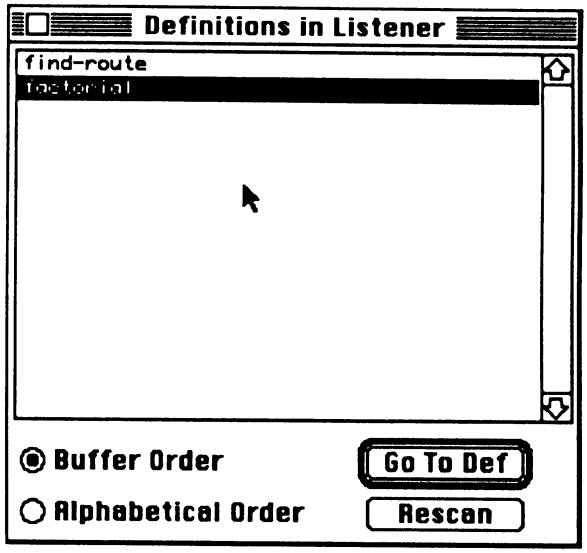


Figure 10

You can then select a function and click on "Go To Def" to rapidly see the source code for the definition (double-clicking on the name will also rapidly get you there). You can view the names by their order in the buffer or alphabetically. You can also rescan the buffer in case it's changed since the last scan. This is part of that "management of complexity" that I talked about earlier (easily getting to definitions in large bodies of code).

"Edit Definition..." is similar; it brings up a dialog into which you type the name of the function to edit. In this case, the file in which the function was defined doesn't even have to be open; Allegro will open it and find the definition for you. In fact,

if the function has been defined more than once and from different sources, another dialog will open indicating the order of the multiple definitions (with the most recent being last) and let you choose which version to edit. I find this feature *very* impressive.

"Apropos..." is for people like me who just can't remember what the name of that function is... I use the extended version of Apropos which is included with Allegro CL 1.1 as source code. It brings up this dialog:

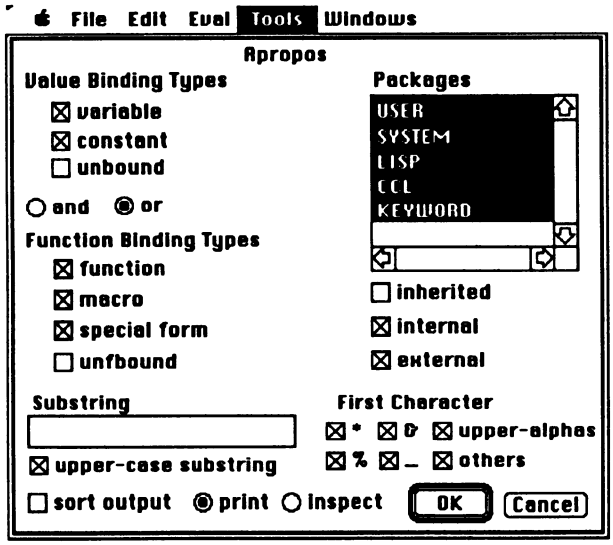


Figure 11

As you can see, you have a lot of choices to make about what Apropos will and will not look for. You can choose what Value Binding Types to look among, and/or what Function Binding Types. You can select the packages to look among, and whether to look for inherited, internal, or external symbols. You can limit your search by first character (thereby only searching things that are, by generally accepted typographical standards, global variables, for example). You can optionally sort the output, and either print it or inspect it.

For all that, what this is really all about is typing some substring into the "Substring" field and letting Allegro list all the symbols that a) meet the criteria specified in the dialog and b) contain the substring that you specified. For example, with the above dialog, here's the list that the substring "button" gets you:

```
:RADIO-BUTTON-CLUSTER   constant value: :RADIO-BUTTON-CLUSTER
:BUTTON2   constant value: :BUTTON2
:BUTTON1   constant value: :BUTTON1
:BUTTON-STRING   constant value: :BUTTON-STRING
:DEFAULT-BUTTON-ENABLED   constant value: :DEFAULT-BUTTON-
  ENABLED
:DEFAULT-BUTTON   constant value: :DEFAULT-BUTTON
:OK-BUTTON-TITLE   constant value: :OK-BUTTON-TITLE
:RADIO-BUTTON-PUSHED-P   constant value: :RADIO-BUTTON-PUSHED-
  P
:DIALOG-BUTTON-ENABLED   constant value: :DIALOG-BUTTON-
  ENABLED
CCL::RADIO-BUTTON-DI-TO-SOURCE   function
CCL::BUTTON-DI-TO-SOURCE   function
*RADIO-BUTTON-DIALOG-ITEM*   value: #<Object #77, *RADIO-
  BUTTON-DIALOG-ITEM*, a CCL::*CONTROL-DIALOG-ITEM*>
*BUTTON-DIALOG-ITEM*   value: #<Object #75, *BUTTON-DIALOG-
  ITEM*, a CCL::*CONTROL-DIALOG-ITEM*>
_BUTTON   constant value: 43380macro
```

Next comes "Inspect," Common Lisp's answer to Smalltalk-80's browser (at least in most respects). Choosing it brings up this dialog:

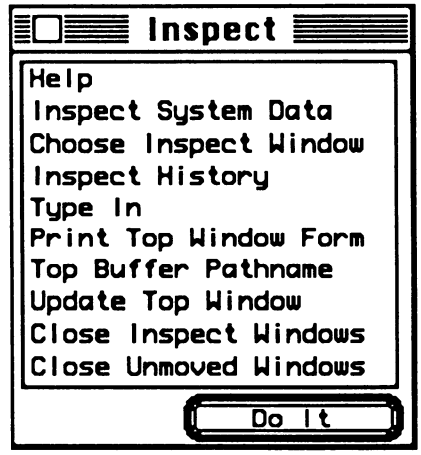


Figure 12

Coral Software has wisely made "Help" the first choice in the dialog. It brings up a text file with some helpful, if somewhat terse, pointers on how to take advantage of Inspect.

The next item, "Inspect System Data," is extremely useful, especially for those just getting into Allegro CL. It shows this window:

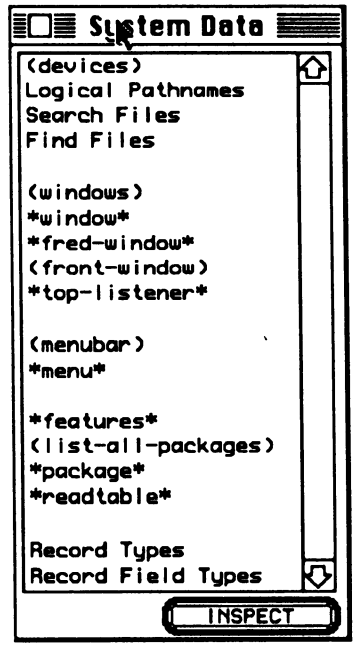


Figure 13

© The Definitive MacTutor, Vol. 4

This window allows you to "inspect" information (and, in some cases, manipulate it) that can be of some value to you and to the system. The first choice in the list is "(devices)," which, as you might expect, tells you about all currently online volumes:

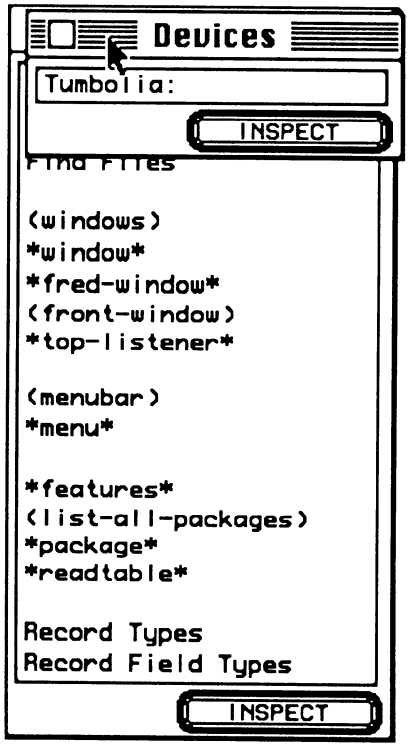


Figure 14

As you can see, I only have one device: my internal hard disk, "Tumbolia." Double-clicking on it gets me this dialog:

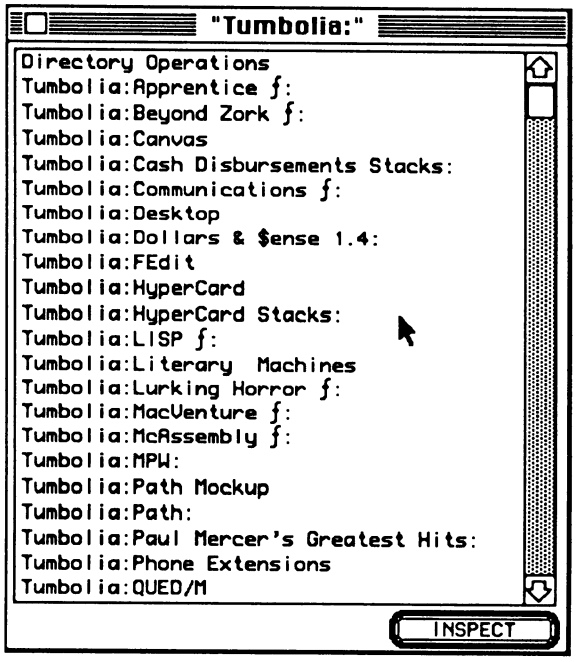


Figure 15

The inspector can perform some Finder-like functions through the Common Lisp file system. There are dialogs for file operations (for those items that don't end with a colon, i.e. those that are files) and directory operations for the ones that do, i.e. are directories. You can copy, delete, load, compile, edit (if it's a text file), and so on, all without leaving Allegro CL.

Other file-related functions available from "Inspect" include the inspecting of logical pathnames (Common Lisp was designed to work on large systems with hierarchical directory structures, e.g. UNIX-based machines and VAXes running VMS. Macintoshes from the 512K Enhanced up fall neatly into this category as well). Common Lisp defines some logical pathnames; Allegro CL provides some more, and you, the programmer, can define still others to make your life easier (much as MPW users can define symbols to refer to complex paths to various files).

A pathname can also be searched for a string, and the pathname being searched can contain wildcard characters, allowing you to define clearly the places to look. "Inspect" can also list all pathnames that match a certain wildcard string.

"Inspect" can also tell you about several system data structures, such as the currently open windows:

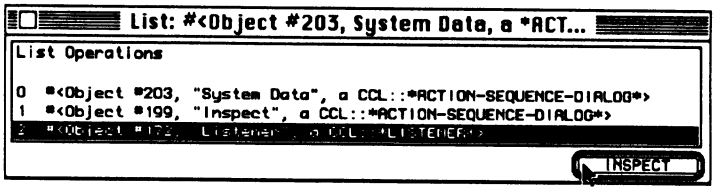
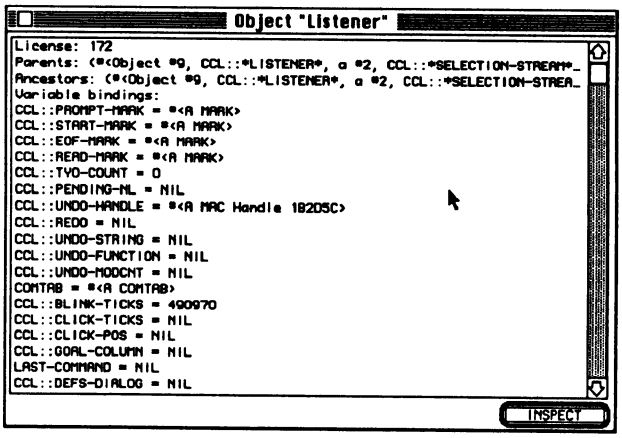


Figure 16

At the time that I double-clicked on "(windows)" in "Inspect," there were only three (at least in that MultiFinder layer)—"System Data," "Inspect," and "Listener." In Allegro CL, you see things referred to as "objects" quite a bit, and they don't mean Lisp data objects, either. Allegro CL is an Object Lisp implementation, and when it says that "Listener" is a "CCL::*LISTENER*," it means that "Listener" is a specific instance of an object called "*LISTENER*" that was defined in the "CCL" package.

Let's see what "Inspect" can tell us about "Listener:"



The window gives us the object's ancestry in the object hierarchy along with the object's variable and function bindings. Values are shown for the variables bound to the object.

The list goes on and gets quite long. Let's move on from "Inspect" and look at some other things.

Back in the "Tools" menu, "Backtrace" is something that can become enabled under certain customizable conditions, such as on an error. Choosing it gets you something like this:

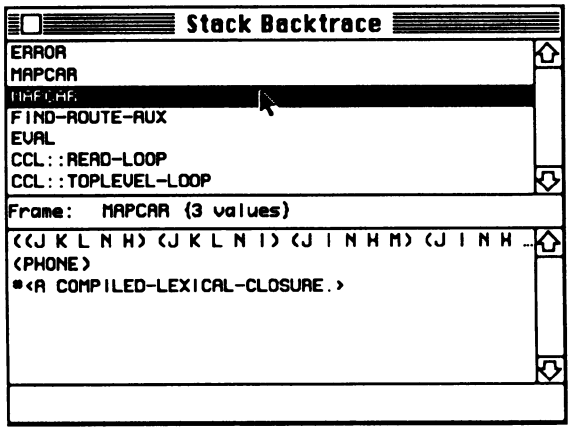


Figure 18

When we got here we had a message indicating that there were too few arguments to MAPCAR (the one immediately preceding the ERROR in the Backtrace). The Backtrace shows us everything that had been evaluated until the error occurred, including Allegro's top-level loop, all the way down to the function that reported the error.

Each function has a frame in which bindings are stored, and those frames can be examined by selecting the function whose frame you're interested in. In the case of the MAPCAR that had the proper number of parameters passed (or so we think—MAPCAR takes optional parameters), the frame shows us three things: a compiled lexical closure that doesn't have a name (because it was written as a LAMBDA expression and passed straight to MAPCAR) and two lists, the first (reading from the bottom up) of which contains the atom "PHONE" and the second of which contains several lists of atoms. Clicking on one of the values shown here will assign that value to the global variable "*DEBUG-VALUE*" so that you may do with it whatever you wish or need to aid in debugging.

"Documents..." is a shortcut to opening a file in the logical pathname "ccl-doc;" which contains several files documenting certain aspects of Common Lisp and Allegro CL in particular, including such Common Lisp esoterica as reader macros, the character set of the language, argument lists for exported symbols in the LISP and CCL packages, and so on. Some of these things are brought up automatically when other functions are performed (for example, a keystroke while the cursor is within most Common Lisp functions will get you its argument list in case you need reminding), but this is a good way to do it explicitly or to get other kinds of information.

Next is "Fred Commands." This is a useful function not only because it can remind you of the standard FRED commands, but it can remind you of non-standard ones as well. I should explain that comment by pointing out that "FRED" stands for "FRED Resembles EMACS Deliberately." The powerful EMACS editor was originally written as a collection of TECO macros, but was eventually made standalone by RMS (Richard M. Stallman), the last of the true hackers from MIT. EMACS has since been commercialized and rewritten (mostly in C) and become almost respectable, but for Allegro CL, FRED remains what EMACS really was: a programmable Lisp editor.

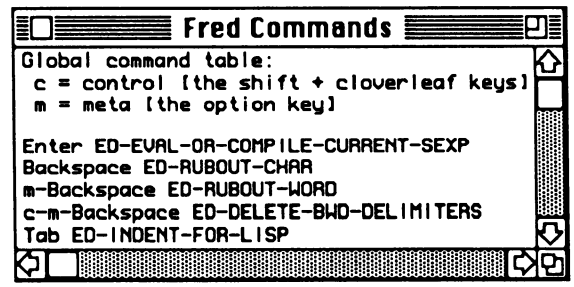
A key idea behind EMACS (and therefore FRED) is that a function can be bound to each and every key on the keyboard. The bindings are kept in a data structure called a COMTAB (*com*mand *tab*le). For most normal keystrokes the function simply inserts the proper ASCII value at the current buffer position, thereby making the keys behave as you would expect.

EMACS expects several other keystrokes to be available, though, thanks to two modifier keys, *control* and *meta*. The Macintosh keyboard historically has had *command* and *option* keys, with *control* being a recent addition not present on all keyboards.

For this reason, Allegro CL offers two modes of operation: Macintosh mode and EMACS mode. In Macintosh mode, the *command* key is the *command* key and *shift-command* is the *control* key. In EMACS mode, this functionality is reversed. In either case, the *option* key serves as EMACS' *meta* key. If the keyboard has a *control* key, it will serve as the *control* key as well (on such keyboards, using Macintosh mode is a good idea).

Since EMACS can redefine the function of the keys, the "Fred Commands" function must determine what those functions are at run-time, and that is what it does. Here is what you see when you choose this function:

Figure 19



This isn't a complete look (the scroll bar is active), but it will give you some idea as to what can be done in EMACS. Pressing the Enter key will evaluate the function "ED-EVAL-OR-COMPILE-CURRENT-SEXP," which in English sounds like "Evaluate or compile the current symbolic expression." In other words, depending upon the context, the s-expression pointed to by the cursor will either be eval'ed or compiled. Note that when EMACS says Enter, it means Enter, not Return.

The Backspace key evaluates "ED-RUBOUT-CHAR," which makes sense. Meta-Backspace does "ED-RUBOUT-WORD," which in light of what the prefix "meta" literally means

also makes sense. Control-meta-Backspace eval's "ED-DELETE-BWD-DELIMITERS," a function that deletes backwards until it finds a matching delimiter (good for removing entire parenthesized expressions).

The function for the Tab key is interesting: "ED-INDENT-FOR-LISP." The implication is that you could bind a function to Tab that would indent for C, or for Pascal, or what have you, and of course, that is the case.

EMACS also supports the concept of a "kill-ring," which you can conceptualize as kind of round-robin clipboards. The most recent kill-ring is the clipboard, as a matter of fact, which brings us back to the "Edit" menu and "Insert Killed String..." This menu item will open a dialog showing the entire kill-ring, and allow you to select any of the available killed strings to be pasted into the current buffer. Using the standard "Paste" menu item will use the most recent string.

The "Print Options..." and "Environment..." choices allow you some control over the Lisp world that you live in. You can choose to pretty-print or not, to print circular structures or not (to do so can hang your machine, however), to use EMACS mode or not, and so on. Note that many of these values can be set in the "init.lisp" file that Allegro CL loads automatically when it is launched. FRED programming can also be done within "init.lisp" for further customization of the environment. Menus can be added and/or changed from "init.lisp" and so forth.

Another great feature of Allegro CL is their single-stepper. The "STEP" function is defined in the Common Lisp standard, but the implementation is left pretty much up to the individual Common Lisp vendor. Allegro CL's stepper is wonderful. Suppose that I have a function called "FIND-ROUTE" that, when given a maze in the form of a list of connected rooms, a starting point, and an ending point, prints the most efficient route(s) to get from start to finish. Let's also suppose that I have a maze whose symbolic name is "CASTLE," with starting point "K" and ending point "PHONE." To step through this function, I would type:

(step (find-route castle 'k 'phone))

and would see something like that shown in figure 20, after a few steps.

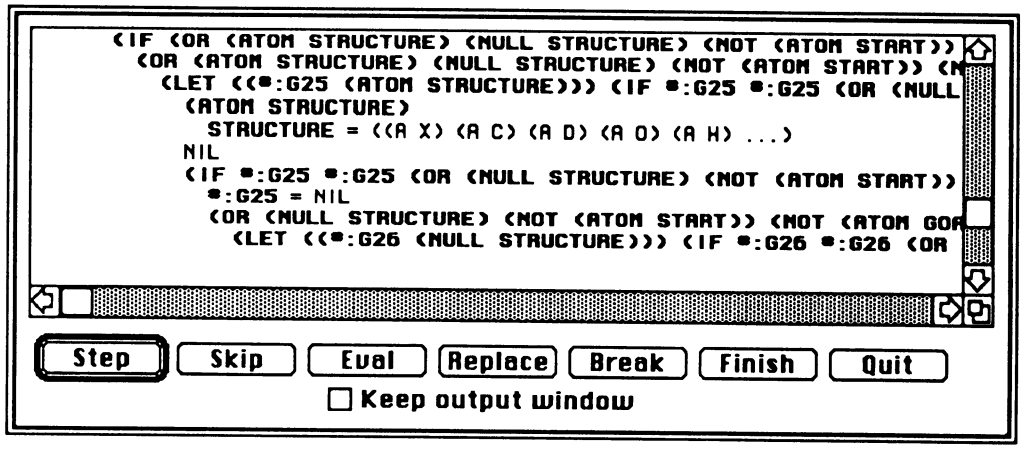


Figure 20

The stepper tells things the way they really are, which can be confusing at times. For example, I practically never use "IF" in Lisp, preferring the more complex, but more flexible "COND" instead. Allegro CL fools me—"COND" is a macro which translates to nested "IF"s, which is why you see "IF"s in figure 20 and none in the Common Lisp code below.

The stepper also evaluates things that are about to be used in a conditional so that you can see why the conditional is about to evaluate to what it is about to evaluate to. In the above example, the code asks "(ATOM STRUCTURE)," so the stepper obligingly mentions that "STRUCTURE" is a list of lists, and so the question "ATOM" is answered "NIL."

The stepper has several functions. The most obvious one is "step." A step in the function can also be "skip"ped. You can "eval" something in the current lexical environment, should you need to. You can also "replace" a step with another expression. You can issue a "break" in case you need to do something like look at a Backtrace. You can "finish" stepping and let the function complete without further ado, or you can "quit" stepping and cancel the current function's evaluation. For any of these choices, you can also keep the current contents of the stepping window after the stepping is over in case you need to scroll through it to see what was going on. As a newcomer to Common Lisp, I have found this stepping power to be indispensable.

Believe it or not, I'm finally finished talking about the Common Lisp programming environment. For the next (and last) few pages, I'd like to point out some differences between MacScheme and Allegro CL as dialects of Lisp rather than as programming environments.

Scheme and Common Lisp both have similar roots. The commonality of them probably stems from basically one person who was highly influential in the creation of both dialects: Guy L. Steele, Jr. Both Scheme and Common Lisp are lexically scoped languages, which sets them apart from older dialects of Lisp, which were dynamically scoped. Both Scheme and Common Lisp lend themselves to solving thorny problems in artificial intelligence, logic, mathematics, and other symbolic tasks. Both dialects were specified with speed in mind (e.g. to be a complete Common Lisp, an implementation must include both an interpreter and a compiler).

Scheme differs from Common Lisp and other Lisp dialects by making functions first-class data objects, i.e. by putting them on equal footing with everything else. LAMBDA expressions in Scheme evaluate to a function, which can be assigned to a variable, passed to MAPCAR or other functions that take a function as a parameter, etc.

Common Lisp doesn't make it quite that simple. If you

intend to pass a function to another function, you'd better call it a function first, even if it is obviously one (because it's a LAMBDA expression or whatever).

Scheme's simplicity allows it to support direct access to some relatively deep internal structures, such as continuations. For us C and Pascal folks, we might want to think of continuations as being analogous to stack frames. A continuation represents the state of a function's execution as it calls other functions and they make their continuations, etc. In Scheme you have explicit access to the current continuation, and can pass it as a parameter to a function that you define. This is a good way to implement escaping from loops, blocks, and procedures, but can also be used to implement fancy control structures such as agendas in artificial intelligence applications. (I cannot tell a lie; the last sentence is a paraphrase of one found in MacScheme+Toolsmith's excellent documentation.)

MacScheme+Toolsmith offers a lot to wander through with their library files and files that define access to virtually all of Inside Macintosh. There's multitasking support, access to machine code definitions for those tricky or time-consuming processes, and standalone application support. MacScheme+Toolsmith may have little in the way of user interface, but it is an excellent example of the elegance and simplicity of Scheme as a dialect of Lisp, and with its Inside Macintosh, multitasking, and machine code support, it packs a *lot* of power into a very small package. To top that off, they haven't forgotten the beginners out there: each copy of MacScheme+Toolsmith includes a copy of Dan Friedman and Matthias Felleisen's *The Little LISPer*, Trade Edition, from The MIT Press. I consider this book to be the beginner's platform for learning Lisp (although the trade edition, which eventually covers such things as the applicative-order Y-combinator and writing a Lisp interpreter in Lisp, does get well beyond the beginning stages).

Scheme also enjoys great popularity in some academic computer-science circles, such as MIT and Stanford.

Common Lisp isn't as conceptually pure as Scheme, perhaps partly because it was designed by committee. It does, however, have a very broad and very rich background, having drawn on ideas from many other dialects of Lisp, including ZetaLisp (the dialect used on Symbolics workstations), MacLisp (developed at MIT for Project Mac—not Macintosh-related at all), InterLisp (found on some Xerox Lisp machines, among other places), and yes, including Scheme as well.

Common Lisp is intended to be a good computational language, with support for everything up to and including complex numbers. In Allegro CL, support for BigNums seems quite speedy compared to MacScheme's BigNum support, and Allegro uses the MC68881 (if one is present) for its floating-point calculations.

Some data types and functions that Scheme lacks are present in Common Lisp. For example, support for sets is part of the Common Lisp standard, as will be seen in the sample code below.

Common Lisp is being widely accepted, partially because it does cover so much intellectual turf, partially because there are many implementations of it available, and partially because a lot of well-known people and companies in the industry had input on it (including some who are in somewhat bitter competition with each other, such as Symbolics, Inc. and LMI).

To sort of highlight one or two of the differences between Scheme and Common Lisp, I've taken a program that was originally written in Scheme and translated it to Common Lisp so that we can look at them side-by-side. The program was originally written in MacScheme by André Marc van Meulebrouck, and is copyrighted by him. He generously posted it to the GEnie network for the rest of us Lispers to enjoy and learn from. Its purpose is, given a data structure that represents a maze as a list of lists of connected rooms, a starting room, and a destination room, to print the most efficient route(s) for getting from start to destination. Here's the Scheme code:

```
; Copyright (c) 1986 by Andre' Marc van Meulebrouck.
; All rights reserved worldwide.
;
;   ┌─────────────────────────────────────┐
;   │  ┌┴┐                 │     ┌┴┐     │
;   │  └┘    M             │ L   └┘   K  │
;   │   │                  │     │      │
;   │   └──┘ └──┘          └──┘ └┘└┘ └┘  │
;   │   │          │       │     │      │
;   │   │  PHONE   │       │  N  │  J   │
;   │   └──────────┘└┘ └┘ └┘ └┘└┘       │
;   │   │         ┌┴┐     ┌┴┐   ┌┴┐     │
;   │ O │         └┘  H  └┘ I └┘   │
;   │   │          └┘ └──┘└──┘└──┘
;   │  ┌┴┐  A    ┌┴┐     ┌┴┐   ┌┴┐     │
;   │  └┘        └┘  D  └┘ E └┘   │
;   │   │          └──────┘└──┘    │
;   │   │         ┌┴┐             │  F  │
;   │   │         └┘   C         │     │
;   │   └──┘ └──┘└──┘ └┘└┘ └┘  │
;   │   │         ┌┴┐             │     │
;   │  ┌┴┐  X    └┘     B       │  G  │
;   │  └┘         │             │     │
;   └──┘└──────────┘└──────────┘└──┘
;

(set! castle '((a x) (a  c) (a d) (a o) (a h) (m  o)
               (x o) (x  b) (b c) (d e) (e f) (f  g)
               (k l) (j  i) (i h) (i n) (n l) (n  h)
               (h d) (m  h) (k j) (m phone)))

;
(define (find-route structure start goal)
  (define (find-route-aux routes)
    (define (goal? routes goal)
      (member #!true
          (appender
           (mapcar (lambda (x)
              (cond ((member goal (last x))
                 (list #!true))
                (else #!null)))
            routes))))
    (define (find-route-io x)
```

```
    (cond ((null? x) #!true)
      (else (display (car    x))
        (newline)
        (find-route-io (cdr x)))))
  (define (goals-only    routes goal)
    (appender (mapcar    (lambda  (x)
            (cond ((equal? (car (last x)) goal)
               (list x))
              (else #!null)))
          routes)))
  (cond ((null? routes) #!null)
    ((null? (goal? routes    goal))
     (find-route-aux (rid-dead-ends
              (make-routes routes))))
    (else (find-route-io (goals-only routes
                  goal)))))
(define (appender x)
  (cond ((null? x) #!null)
    (else (append (car x)    (appender (cdr x))))))
(define (make-routes routes)
  (define (make-routes-aux route)
    (define (choices room)
  (define (set-difference    set1 set2)
    (appender (mapcar (lambda (x)
                (cond ((member x set2) #!null)
                  (else (list x))))
              set1)))
  (define (extricate x)
    (cond ((null? x) #!null)
      ((member (car x) (cdr x))
       (extricate (cdr x)))
      (else (append (list (car x))
                (extricate (cdr x))))))
  (set-difference (extricate (appender (mapcar (lambda (x)
                                (cond ((equal? (car x) room)
                                   (cdr x))
                                  ((equal? (cadr x) room)
                                   (list (car x)))
                                  (else #!null)))
                              structure))) route))
    (appender (mapcar    (lambda (x)
            (list (append route
                  (list x))))
          (choices (car (last route))))))
  (appender (mapcar (lambda (x)
            (make-routes-aux x))
           routes)))
(define (rid-dead-ends routes)
  (appender (mapcar (lambda (x)
            (cond ((null? (car (last x)))
                 #!null)
                (else (list x))))
           routes)))
(define (last x)
  (cond ((null? x) #!null)
      (else (list (car (reverse x))))))
  (cond ((or (atom? structure)
         (null? structure)
         (pair? start)
         (pair? goal))
     #!null)
    ((equal? start goal) goal)
    (else (find-route-aux (rid-dead-ends
                  (make-routes (list (list start))))))))
```

There it is, in all its splendor. Notice the heavy use of implicit iteration via MAPCAR. Also notice the use of LAMBDA expressions to pass functions to MAPCAR on the fly, without the bother of defining them and assigning them to a symbol and whatnot. Also note that what the LAMBDA expressions evaluate to is already a function ready to go to MAPCAR. No additional sugar is necessary.

Here's my Common Lisp translation of the same code:

```
; Copyright (c) 1986 by Andre' Marc van Meulebrouck.
; All rights reserved worldwide.
; Common Lisp translation by Paul F. Snively
;
;  ____________________________________
;  |  _|_                |     _|_     |
;  |  __     M           | L    __  K  |
;  |   |                 |      |      |
;  |   |___| |____       |__| |__|__| |_|
;  |   |     |           |     |       |
;  |   | PHONE |         |  N  |   J   |
;  |   |_______|__| |__|  |_| |_|_      |
;  |   |          _|_      _|_   _|_    |
;  | O |          __  H    __ I  __     |
;  |   |          |___| |____|_____|____|
;  |  _|_  A     _|_        _|_   _|_    |
;  |  __         __  D     __  E  __     |
;  |   |         |_________|_____|        |
;  |   |         _|_                |  F  |
;  |   |         __     C            |     |
;  |   |___| |___|_____| |____|__| |_|
;  |   |         _|_                |     |
;  |  _|_  X     __        B       |  G  |
;  |   |         |                 |     |
;  |___|_________|_________________|_____|
;
(setq castle '((a x) (a    c) (a d) (a o) (a h) (m    o)
                (x o) (x   b) (b c) (d e) (e f) (f    g)
                (k l) (j   i) (i h) (i n) (n l) (n    h)
                (h d) (m   h) (k j) (m phone)))
;
(defun find-route (structure start goal)
  (defun find-route-aux    (routes)
    (defun goalp (routes goal)
      (member T
          (appender
           (mapcar (function (LAMBDA (x)
                    (cond ((member goal (last x))
                       (list  T))
                      (T NIL))))
                routes))))
    (defun find-route-io (x)
      (cond ((null x) T)
        (T (print (car x))
          (find-route-io (cdr x)))))
    (defun goals-only (routes goal)
      (appender (mapcar    (function (LAMBDA (x)
                    (cond ((equalp (car    (last x)) goal)
                       (list x))
                      (T NIL))))
            routes)))
```

```
   (cond ((null routes) NIL)
     ((null (goalp routes goal))
      (find-route-aux (rid-dead-ends
                (make-routes routes))))
     (T (find-route-io (goals-only routes
                       goal)))))
 (defun appender (x)
   (cond ((null x) NIL)
     (T (append (car x) (appender (cdr x))))))
 (defun make-routes (routes)
   (defun make-routes-aux (route)
     (defun choices (room)
    (defun extricate (x)
      (cond ((null x) NIL)
        ((member (car x) (cdr x))
         (extricate (cdr x)))
        (T (append (list (car x))
                (extricate (cdr x))))))
    (set-difference (extricate (appender (mapcar (function
(LAMBDA (x)
                                  (cond ((equalp (car x)room)
                                     (cdr x))
                                        ((equalp (cadr x) room)
                                          (list (car x)))
                                            (T NIL))))
                                 structure))) route))
     (appender (mapcar    (function (LAMBDA (x)
                       (list (append route
                                (list x)))))
              (choices (car (last route))))))
   (appender (mapcar (function (LAMBDA (x)
                    (make-routes-aux x)))
                routes)))
 (defun rid-dead-ends (routes)
   (appender (mapcar (function (LAMBDA (x)
                    (cond ((null (car (last x)))
                       NIL)
                      (T (list x)))))
                routes)))
 (cond ((or (atom structure)
        (null structure)
        (not (atom start))
        (not (atom goal)))
    NIL)
    ((equalp start goal) goal)
    (T (find-route-aux (rid-dead-ends
                (make-routes (list (list start))))))))
```

It looks much the same, doesn't it? If you look closely, you'll see a lot of minor syntactical differences, such as the use of "T" instead of "#!true." More importantly, however, you'll notice that the definition of "set-difference" is missing, since Common Lisp already has it, and you'll notice the "(function...)" around the LAMBDA expressions, because Common Lisp wants it that way. Also, there's no "pair?" in Common Lisp, so the question was "What is he really looking for?" and the answer was "He wants to make sure that the start and destination are atoms." Hence the "(not (atom...))" lines near the end.

Someone who is a bit more of a "Common Lisp Rambo" than I am at the moment might think in terms of restructuring this program so that you can eliminate the definition of "appender" and replace the numerous "(appender (mapcar...))" invocations with "(mapcan...)" ones. The problem with that is that "MAPCAN" is a destructive operator (one that performs surgery on lists, rather than copying them) that can exhibit some pretty weird side-effects if you aren't careful with it. The optimization is left as an exercise for the reader.

Well, that wraps up this look at two excellent Lisp systems for the Macintosh. I expect to see both of these products evolve over time. MacScheme has been evolving, and has become a product that we Scheme fans can really appreciate, especially with its good documentation, large amounts of source code to tinker with and learn from, and its support of the Toolbox, multitasking, machine code interfacing, and standalone applications.

Allegro CL is remarkable as the first full Common Lisp for any mass-marketed microcomputer, and for its extensive and powerful user interface. It also offers excellent Toolbox access and several interesting example programs, including color examples and Cosell, a Common Lisp spreadsheet. The lion's share of the documentation *is Common LISP: The Language*, which in Common Lisp circles is referred to as "The Silver Bible." A copy is included with the product. Also included is a copy of *Common LISP: The Index*, by Coral's own Rosemary Simpson, which attempts to improve (and succeeds at improving) upon the index that is in the book. The remaining documentation is a somewhat terse description of Allegro CL specifics that would benefit from a less dry, "just the facts, ma'am," more lively style and from more examples. Issues remaining to be resolved are multitasking, interfaces to other languages, and standalone applications. However, Coral has clearly positioned this product as being in line with the likes of Symbolics' work, especially with the introduction of Allegro Flavors and the promised support of CLOS and Common Windows.

I'm lucky—I've got 'em both. Other folks might not be so lucky. If you're looking for a Lisp system, both are recommended highly, albeit for different reasons. Scheme fans, MacScheme+Toolsmith will let you do just about anything you want, up to and including creating a standalone application that you can then sell or give away or whatever royalty-free.

If you use Common Lisp professionally and have been looking for a real Common Lisp for your Macintosh, Allegro CL is it. It's 100% bona-fide genuine Common Lisp, not some little subset. Speaking of which, if you're looking for a good, inexpensive subset, keep your eyes on Coral, who are talking about doing "Coral Lisp," a subset of Allegro CL, for about $100.00. In fact, it should be out by the time you read this. This sounds like it'll be a great entry-level system for the first-time Lisper. Maybe Coral will take a hint from Semantic Microsystems and bundle *The Little LISPer* with their introductory Lisp.

MacScheme+Toolsmith 1.0
$395.00
Semantic Microsystems, Inc.
4470 S.W. Hall, Suite 340
Beaverton, OR 97005
(503) 643-4539

Allegro CL 1.1
$599.95
Coral Software, Inc.
P.O. Box 307
Cambridge, MA 02142
Orders: (800) 521-1027
Tech Support: (617) 547-2662

Learning Lisp

Lisp Objects: Ford vs Chevy

Didier Guillon
Grenoble, France
MacTutor Vol. 4 No. 4

Didier Guillon is a former restaurant manager, currently in the fourth year of a Computer Science program at Grenoble University in France. He is a member of AAAI and works in artificial intelligence as a consultant at the Medical University of Grenoble's department of BioStatistics. He is also president of Mac Alpes, a club of Macintosh developers in Grenoble and has been using the Mac for two years.

Welcome to this new column. When I first started Macintosh programming, MacTutor and the contributing editors were there and helped me since issue number one. Now I think it's time to share with you the knowledge I gained with the "oldest new language". So welcome to symbolic computation... you have a problem? No, don't be frightened, I have been so pleased when I discovered ABC' s of C that I wanted to introduce you to the Lisp world in a gentle way. The technique I will use may sometimes be criticized by expert Lisp programmers...but do they remember their first steps?

Today I will present the object idea with a very dumb example and "how to" on the Macintosh. MacApp and MPW Pascal are there to let us remember that object programming becomes a reality day after day. But, since Smalltalk, no other language than Common Lisp did it with better efficiency.

My Kingdom for an Object

But what is an object ? To the outside world, an object is an entity. But for us programmers, an object is some private memory with a set of operations. An object can be a number, a square, a text editor. An object which is a square knows a lot of information about itself: how to calculate the area of the square, the center of the square... When we want the object 'square' to carry out such a calculation, we send him a message (we say: to invoke). But we just tell him which operation we want, not how to operate. (If he's smart, he already knows that!)

When a set of objects represents the same kind of information, we implement a class with a data structure and the procedures that operate on this data structure. For example, a class implemented for a rectangle would know how a specific object, called an instance, would carry out the operation rectangular area. This instance has private variables called instance variables and the function that will operate on the instance of a particular class is call a method. We just defined in fact the fundamental concepts of object programming: data abstraction, information hiding (not to tell "how to") and generic operation. But all this is just theory, so let's start with our dumb example. We will use the **ExperCommonLisp (ECL) from ExperTelligence™**, who since the last review of this language in MacTutor, did a lot of improvement and removed many bugs.

First, let's create a class for 'cars' (note: we wrote cars to avoid the "car" word which is a Lisp reserved word).

```
(defclass (cars object)
  (IVS (color 'blue)
       (price 0)))
```

Defclass contains a list with the name of the call and the name of the superclass. In ECL all the classes come from a special class called object. In fact object is a root class from which all classes are derived. The IVS stands for instance variable and as we saw previously, will manage to store the IVS in each instance. You note the quote before blue? In Lisp every elementary object, called an atom, is evaluated. This evaluation returns a function name or the value attached to this specific atom. So, to prevent evaluation, we "quote" the atom which becomes itself a value and is attached to the instance variable color. This 'quoting' is not necessary for a numeric value. Then we define the method (using defmethod followed by the name of the class and the name of the method itself):

```
(defmethod (cars small) ()
   (setq price 800)
      (format T "A small car is just fun for ~a dollars.~%" price))
```

This method is doing two things: setting price to 800 with setq, a Lisp word which will not evaluate the following word, price, and sets its value to 800 (for Pascal programmers it's an assignment). Then it prints a message (good old Fortran world !!!). You are now able to trace by yourself what the second method does:

```
(defmethod (cars race) ()
      (setq color 'red)
      (setq price 20000)
      (format T "A wonderful race car whose color is ~a.~%" color))
```

Now we link up what we define to this first class: we compile and define an instance (we will represent the return value by ->):

```
(setq mylcar (cars 'new))
-> #<CARS INSTANCE $5402341A>
```

"mylcar" is the instance which holds a copy of the class 'cars'. Let's invoke the instance in various ways :

```
(mylcar 'race)
-> A wonderful race car whose color is RED.
```

© The Definitive MacTutor, Vol. 4

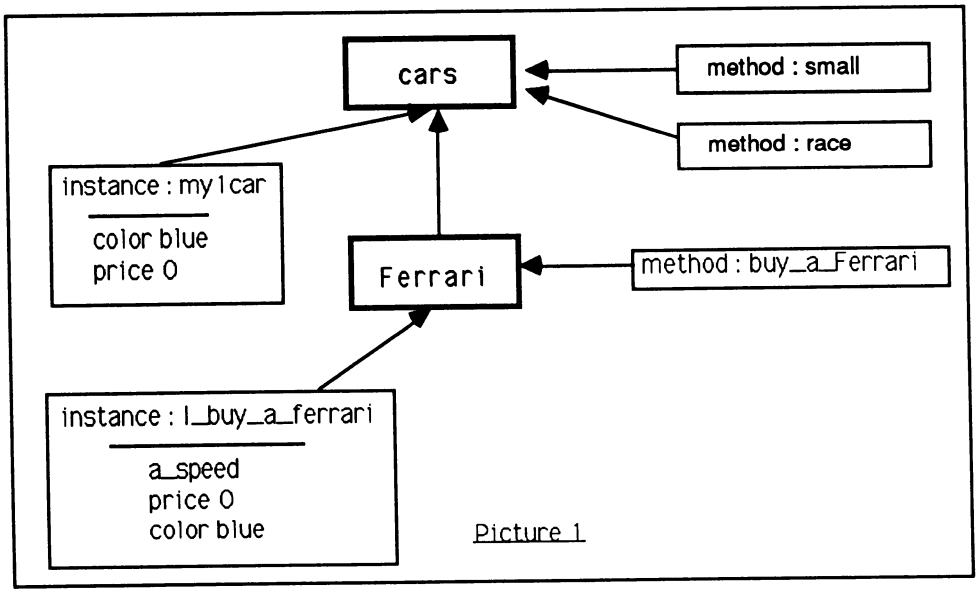


Fig. 1 Our cars example illustrates basic concepts

We can see easily that I_buy_a_Ferrari inherits from its superclass 'cars' all the methods and instance variables. We will not cover here some other concepts like class variables, super....

So now that we have an idea on how objects live, we'll have a look at ECL. In ECL, the object idea has been expanded to the Macintosh Toolbox.

Before going on to the Macintosh toolbox, we have a look at another object expression, metamethod. A metamethod is a procedure which is invoked by sending an appropriate message to a class. We explain metamethod because on the Macintosh we create a NEW metamethod which sets up the required data structure. When invoking a specific metamethod, the instance of a particular class will have instance variables which will correlate (and with the same name) to the equivalent fields in the Pascal data structure. An example from QuickDraw:

```
(setq yourRect (Rect 'new 30 40 250 280))
```

We just created the instance call 'yourRect' with a set of parameters. If we send it a message, we can access these instance accessors by the same name as in the Pascal structure : top, left, bottom, right.

```
(yourRect 'bottom)
-> 250
```

or change their value :

```
(yourRect 'left 60)
-> 60
```

We can also draw, calling Toolbox procedures with the instance 'yourRect' :

```
(!PaintRect yourRect)
```

Note that trap names are prefixed by an exclamation mark. We are now ready to start programming our first minimum shell in ECL... but that's for next month.

For those interested in following me, and starting Lisp, I recommend the very nice and easy book of Touretzky "A Gentle Introduction to Symbolic Computation". For the others, obviously, Lisp 2e by Winston is a must. I intend to present the next coming months various aspects of Lisp programming in Artifi-

```
(mylcar 'small)
-> A small car is just fun for 800 dollars.

(mylcar 'price)
-> 800

(mylcar 'price 200)
-> 200

(mylcar 'price)
-> 200
```

We can see the graphic image of the object we define, including the next subclass we will create:

Here is the subclass Ferrari:

```
(defclass (Ferrari cars)
      (IVS a_speed))

(defmethod (cars buy_a_Ferrari) ()
      (setq a_speed 200)
      (self 'race)
      (format T "But when you drive at ~a miles... Whoo !!!!~%"
a_speed ) )
```

Same as the 'cars' object, but note the "self" word which is used to invoke the method race from the method buy_a_Ferrari. As previously, we can say:

```
(setq I_buy_a_Ferrari (Ferrari 'new))
-> #<FERRARI INSTANCE $540233AA>

(I_buy_a_Ferrari 'buy_a_Ferrari)
-> A wonderful race car whose color is RED.
    But when you drive at 200 miles... Whoo !!!!

(I_buy_a_Ferrari 'price)
-> 20000

(I_buy_a_Ferrari 'color)
-> RED
```

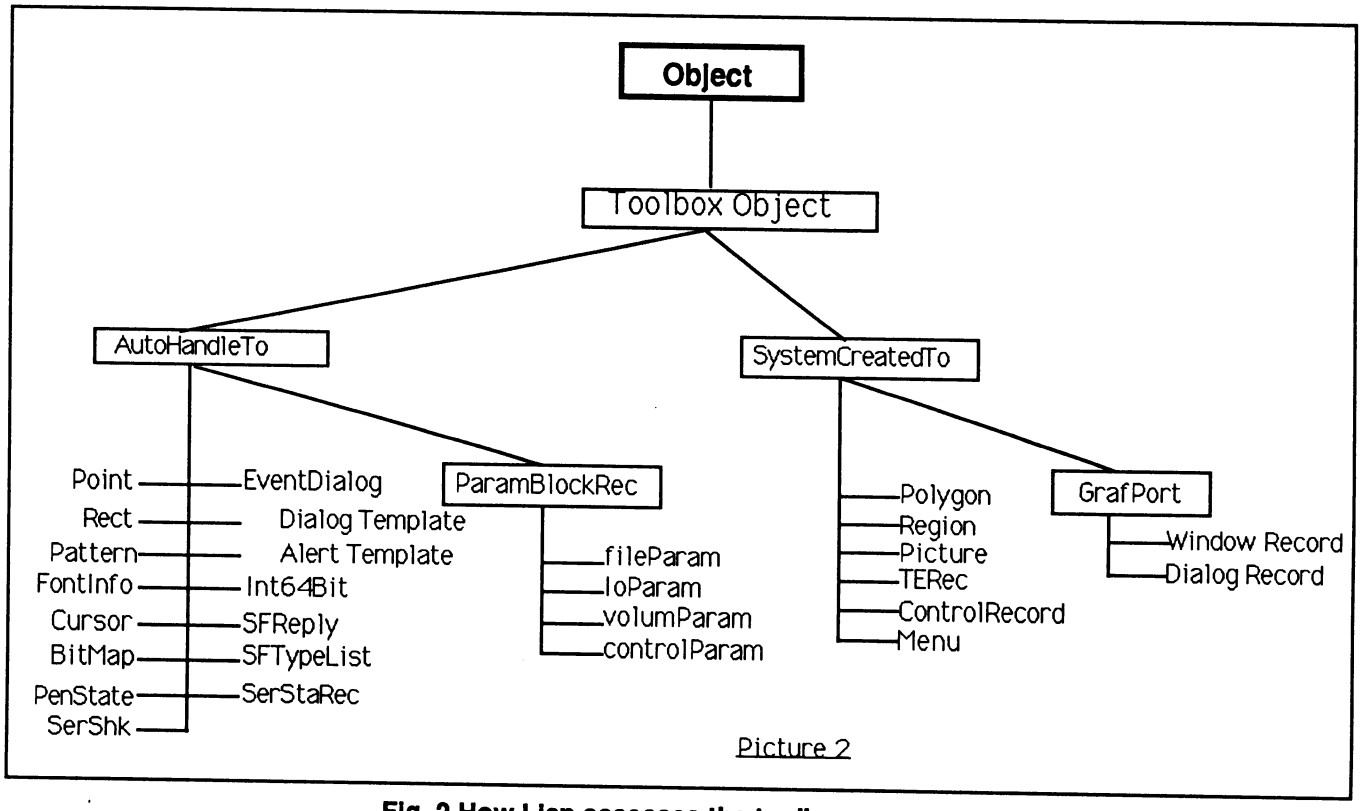


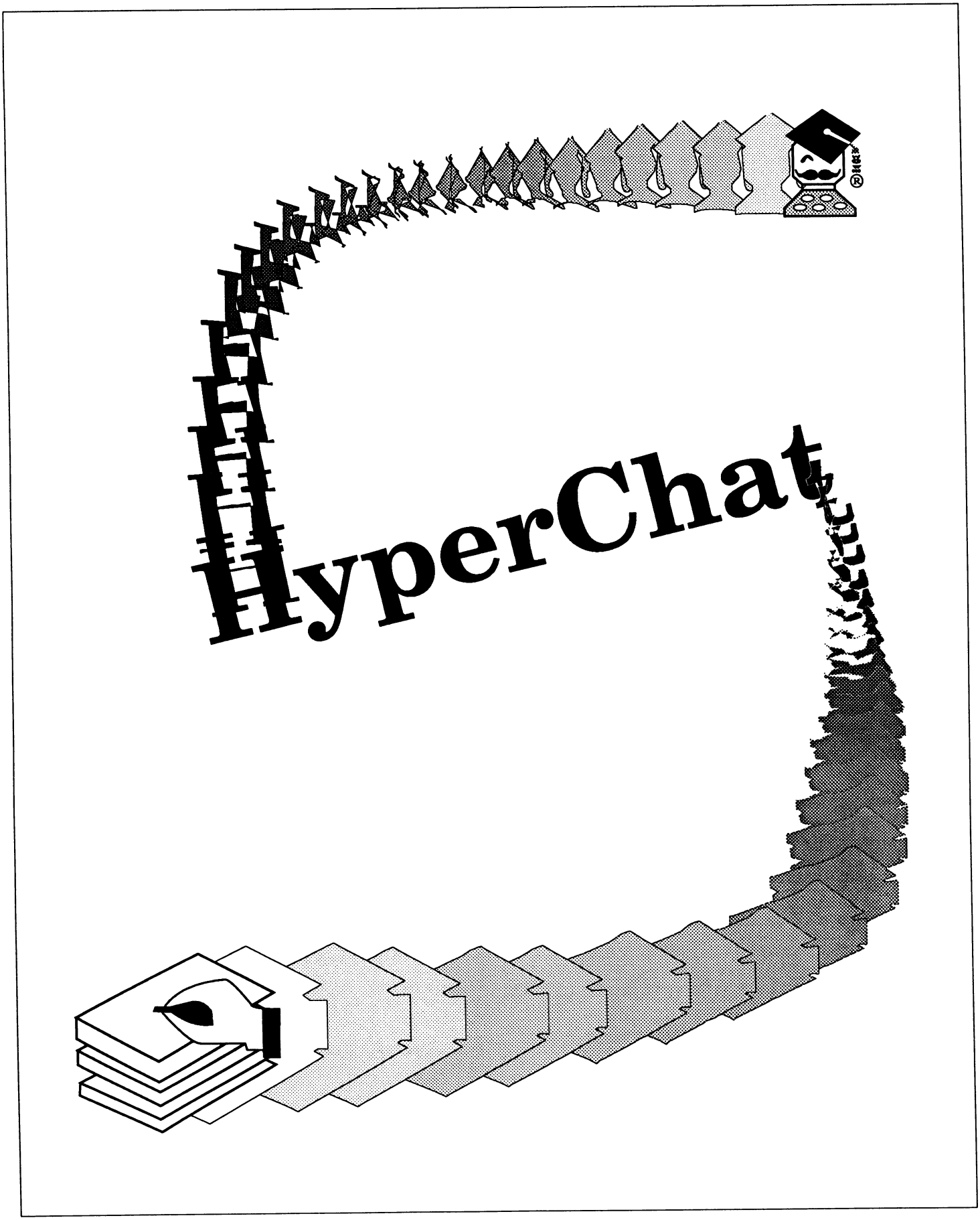
Fig. 2 How Lisp accesses the toolbox

cial Intelligence, but also use some special languages as OPS 5, SmallTalk, Humble, Prolog (or why not Nexpert, just to let us dream...). If you have any suggestions, please contact me :

Didier Guillon
8 Impasse Saint Jean
38240 Meylan
France.

The complete program :

```
(defclass (cars object)
      (IVS (color 'blue)
       (price 0)))

(defmethod (cars small) ()
      (setq price 800)
      (format T "A small car is just fun for ~a  dollars.~%"
price))


(defmethod (cars race) ()
      (setq color 'red)
      (setq price 20000)
      (format T "A wonderful race car whose color  is ~a.~%"
color))

;**********Examples*****************

;(setq mylcar (cars 'new))
;#<CARS INSTANCE $5402341A>
;(mylcar 'race)
;A wonderful race car whose color is RED.
;(mylcar 'small)
;A small car is just fun for 800 dollars.
;(mylcar 'price)
;800
;(mylcar 'price 200)
;200
;(mylcar 'price)
;200

;**********Second class**************

(defclass (Ferrari cars)
  (IVS a_speed))

(defmethod (Ferrari buy_a_Ferrari) ()
   (setq a_speed 200)
   (self 'race)
      (format T "But when you drive at ~a miles ...  Whoo
!!!!~%"                                          a_speed))

;***********Example****************

;(setq I_buy_a_Ferrari (Ferrari 'new))
;#<FERRARI INSTANCE $540233AA>

;(I_buy_a_Ferrari 'buy_a_Ferrari)
;A wonderful race car whose color is RED.
;But when you drive at 200 miles... Whoo !!!!

;(I_buy_a_Ferrari 'price)
;20000

;(I_buy_a_Ferrari 'color)
;RED
```



© The Definitive MacTutor, Vol. 4

HyperChat™

New HyperCard Products

Fred Stauder
Contributing Editor
MacTutor Vol. 4 No. 5

HyperChat is a new section of MacTutor which will bring you new hypertalk programming techniques and lead you into the realm of "User Friendly Programming".

Each month we aim to inform you of the latest developments in the Hypercard Community, such as new products, ScriptTips, XCMD's and Articles. I am sure you will enjoy HyperChat as much as we enjoy bringing it to you.

Is Hyperware Vaporware?

At the introduction of Hypercard there was much enthusiasm from the Mac Community, also an outcry from the developers who saw some of their plans dissrupted. What was Apple trying to do? Is this system software? Did people scream out when MPW was introduced? Of course not, it was a tool to be used by developers. Apple has positioned Hypercard the same way, as a tool. There is a difference though in that Hypercard is the tool "for the rest of us".

Are you having a DejaVu? A lot of people laughed at the Mac when it was introduced. How can this toy do real computing? How can you do a real application with Hypercard? Now we have seen an explosion of Stacks for Hypercard. Take a look at them; a good proportion are garbage. Look at many of the early programs on the Mac, were they not also garbage? I believe it is only a "teething"problem. Now take a look at two professional programs written in Hypertalk, Buisness Class, and Focal point. There is a market for Professional Hypercard Stacks, there is no market for garbage. The sales of these stacks has been remarkable. People are finding them very productive to use in their daily lives. Hypertalk is not a passing phase, just as the Mac wasn't, it will continue to evolve and grow. Why should programmers be scared of this new thing? Hypercard will give you more oppotunities, it will allow you to do some "look and feel" tests before you write code. You can also do quick jobs that would just take too much time to write in another way for them to be economical. It is also very useful to doccument and spec out projects. A famous Australian expression is; "you'll never know unless you 'ava go".

If you think it too beneath you to program in Hypertalk look at the people who use it, including Bill Atkinson the visionary who wrote Hypercard. Many of us are caught up in the investment time it takes to learn a language, so to go to another one we percieve as a loss. Our aim should be to hack concepts and ideas instead of syntax.

Hypercard at MacWorld: San Fransisco

Hypercard made its commercial debut at MacWorld, with the introduction of dozens of commercial stacks, books and publications.

One of the hilites of the show was seeing Bill Atkinson demonstrate Hypercard. It was such a joy to watch the audience come alive with such enthusiasm. Bill was lucky though because he had the help of the youngest HyperHacker, his daughter Laura (you probably have seen her digitised photo in the slide show stack that came with Hypercard).

Laura would yell out the command keys to her dad to make him get to the good bits faster.

After Bill's talk I spoke with Amanda Goodenough who wrote "Inigo Gets Out", "Inigo Takes a Bath", and "Your Faithful Camel"(for those of you who have not seen them these are a great example of childrens stacks). Amanda told me how young kids are taking to Hypercard. Using their active imaginations and the multi dimensions of Hypercard to create wonderful things. [One of the best ways to test your software is to let children try it. They will quickly point out conceptual inconsistancies.]

Later on Laura Atkinson was using a MacRecorder to input sound into Hypercard. It was such a joy to see a young child doing, without much assistance, what MIS managers with their multi million dollar computers, can't do.

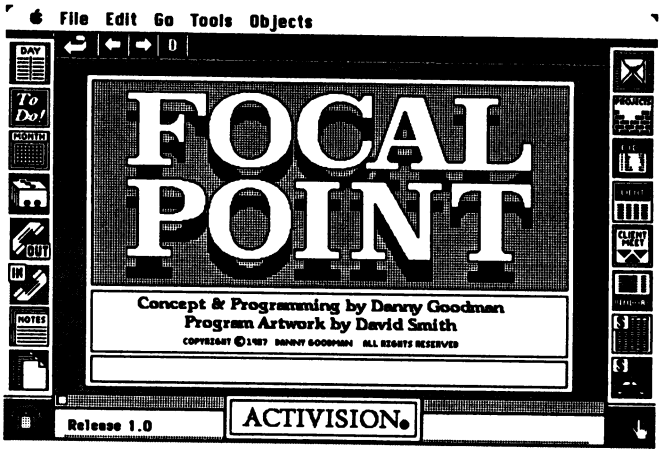
A lot of people were talking about the limitations of Hypercard. Well I suppose everyone has forgotten that Apple did listen to the users about the limitations of the Mac.

I think the Mac II is a good response to people's needs. My advice is good products take time; be patient.

There were too many things to mention them all so I will give examples of what I feel are the most innovative.

Focal Point and Buisness Class

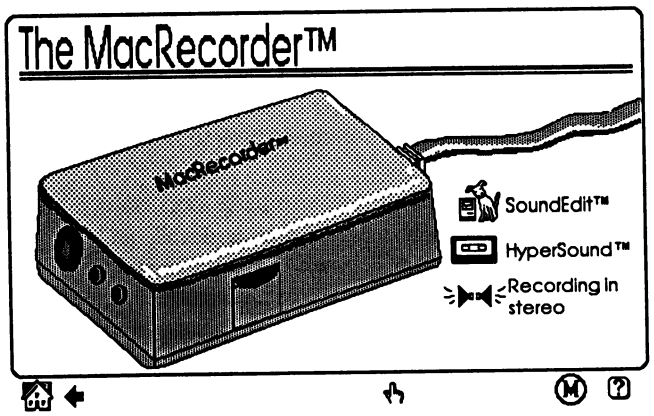
If you haven't seen these stacks; you should. Danny Goodman did a remarkable job developing these stacks during the time of quite a few revisons of Hypercard. I won't go into details, but I would like to point out that these professinal stacks made a big impact at MacWorld SF.



© The Definitive MacTutor, Vol. 4

MacRecorder

Farallon has released the MacRecorder with two programs, SoundEdit and HyperSound. These two programs are very good examples of what should be done to make a good interface; clean, simple, and a pleasure to use. Written by Steve Capps and the Farallon team. HyperSound lets you record a sound and place it directly into a stack, ready to play. (see HyperTests)

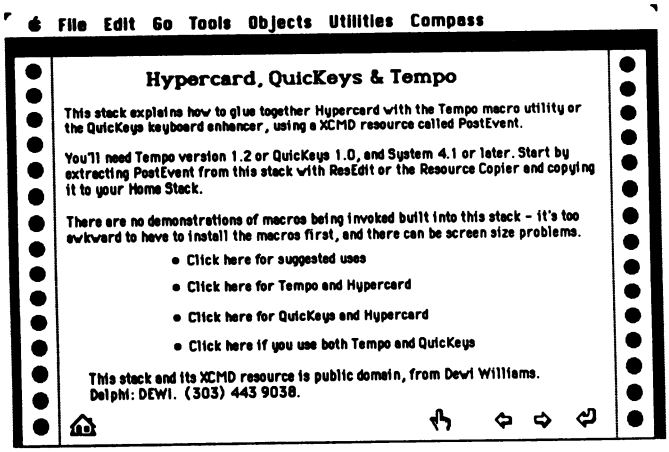


Tempo II

This new version of Tempo is a much more powerful version of the original.

With an XCMD called PostEvent written by Dewi Williams you can call a Tempo II macro directly from hypercard, run the macro, and return control to hypercard. This allows you to automate things that you always wanted to do. Such as issuing click commands for dialog boxes. Hypertalk won't allow you to invoke desk accessory menu items via doMenu. This can be done now with a simple macro.

PostEvent can also be used with Quickeys.

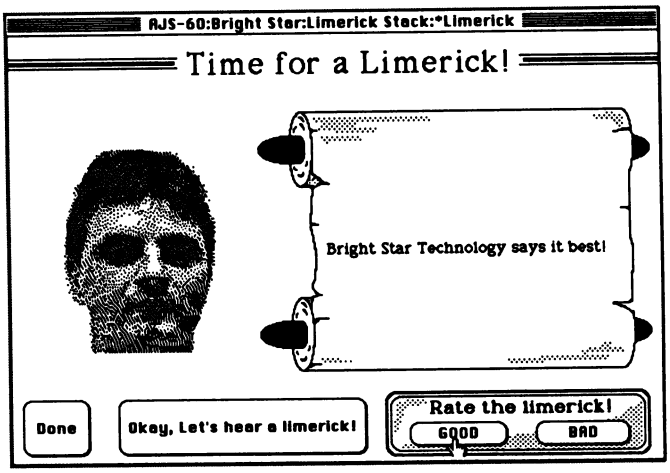


Bright Star

Some people may have missed this item at MacWorld Expo in SF and if you did it is a pity.

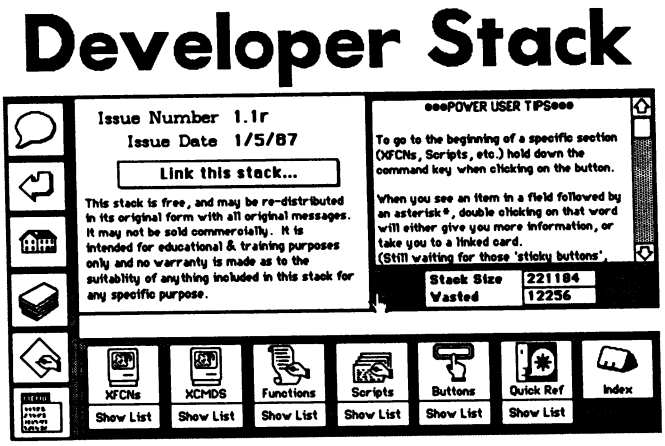
It is based on Bright Star Technology's real Time Animation and Vivification Engine, the RAVE, and it's RAVEL language. It drives a face in Hypercard that will mouth the words from ASCII text, and simultaneously speak the text. The power of this language is amazing. The images are digitized (the one in the figure is of yours truely) with a video camera.

The number of images used for the phonemes can be as few as 5 images or for greater precision 50 or more. Ravel allows color images as well, and is language independent. For example if you can have French text typed in and you will see your face move to French words. Bright Star have produced some applications for teaching purposes. The potential of such technology holds a bright future for Bright Star.



Hottest Development Stack

Developer Stack 1.1, written by Steve Drazga, contains, XCMD's, XFCN's, Scripts, and a bunch of other goodies. It is designed to be a resource for Hypercard development. The interface is very clean and not cluttered like some other Hypercard Development tools (In most cases when you design Stacks; more is not better). Steve is developing an upgrade stack that will update the Developer Stack. This Stack is available on most Bulletin Boards and from user groups.



Most Useful XCMD

ResCopy written by Apple's XCMD Guru Steve Maller is a utility that allows you to copy resources from one file to another

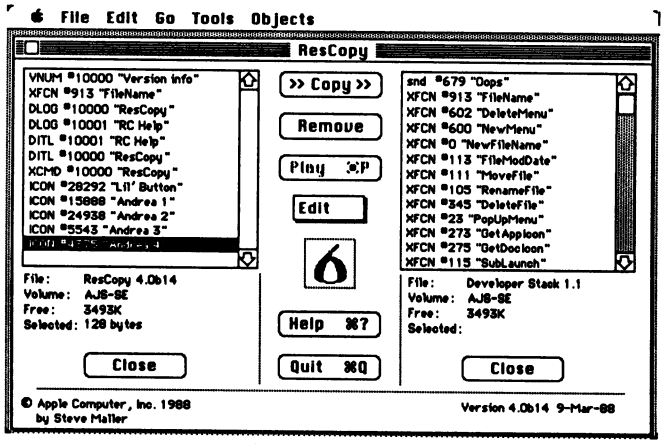
directly through a dialog box or by using scripts.

ResCopy works in a similar fashion to the Font/DA mover. You call install ResCopy into your home stack typing ResCopy into the message box brings up a window (see figure 6). You can copy, remove, rename, renumber resources, and play sounds. One really useful feature will allow you to convert different snd resources to Hypercard format.

ResCopy allows direct resource copying from scripts. This is useful, for example, if you want to install a cursor or other resource into the Home Stack. The copying takes the form:

ResCopy fromFile, toFile, "XCMD", 10000.

This XCMD is a must for Hypercard developers.



—ScriptTips

The first ScriptTip allows you to edit the script of a stack, Background, Card, Field, or Button. You can put this script into the Home Stack so it can be used in all stacks. To edit a button or a field hold the mouse button down over a field then move the mouse up with the button still depressed. The script editor will then be envoked. To edit a stack script choose part of the card that does not have any buttons or fields then move the mouse to the top of the card, (or past it if you have a large screen) while holding down the mouse button. To edit the background script, move to the left of the card with the mouse button down, and to edit the card script move right. I have found it helps when you are debugging and it will work in most stacks.

```
——— AutoScriptEdit by Fred Stauder ———

- Put this Script into your Home Stack
- To use keep mouse down over Button or Field and move mouse
up
- To edit Bkground Script keep mouse down - move to very left
of card
- Card Script - move right
- Stack Script - move up

on mousestilldown
  set cursor to 4
  put char 6 of the target into C6
  if C6 is quote then AutoScriptEdit
  if C6 is "i" then AutoScriptEdit
  if C6 is quote then exit mousestilldown
  if C6 is "i" then exit mousestilldown
  if the mouseV <item 2 of rect of target then edit script of
target
  end mousestilldown

  on AutoScriptEdit
    if the mouseH < 2   then edit script of this bkgnd
    if the mouseH > 510 then edit script of this card
    if the mouseV < 2   then edit script of this stack
  end AutoScriptEdit

  - Non-Comercial use only - All Rights Reserved
  - Fred Stauder, Hypercard Editor, (HyperChat) MacTutor March
1988
```

This second ScriptTip is a convenient method to bypass a card that is not normally important for everyday use, such as an about card. It is also a good place to hold information that the user does not have to see. You can easily set a flag to access the card when you want to get information from it.

```
——— HideCard by Fred Stauder ———
- This script allows you to "hide" cards
- For example an about box card
- It does not completely hide the card it just passes it
- Put it in to the Card Script

on opencard
  if the optionkey is down then exit opencard
  - put in any type of flag eg:
  - if FlagUnLock is true then exit opencard
  if the number of this card >the number of recent card ¬
  then go to next card
  if the number of this card <the number of recent card ¬
  then go to previous card
end opencard

  - Non-Comercial use only - All Rights Reserved
  - Fred Stauder, Hypercard Editor, (HyperChat) MacTutor March
1988
```

—HyperTest

Farallon's MacRecorder, HyperSound, and SoundEdit

The MacRecorder is a sound digitizer that is slightly larger than a mouse. it draws is's power from the serial port, so there is no power supply needed. It has a built in microphone, mic jack, iine jack, and volume control. Using two units you can record in stereo. Beeps sound incredible going from one speaker to the other. The MacRecorder is packaged in a handy bookshelf box containing, a MacRecorder, cables, a great manual which includes theory of sound, and three disks.

Hypersound is a hypercard stack that looks like a tape recorder and best of all acts like one it is a very simple stack and a good example of a clean interface. (see figure 7 next page)

First you check the input level and adjust the volume control on the recorder. Then all you have to do is hit the record button and the sound will be recorded as a "snd " resource. There is a button that allows you to copy the sound to a stack. You choose the stack, the snd is copied and a button is created that plays the sound. There is a button that launches the second program of the package; SoundEdit.

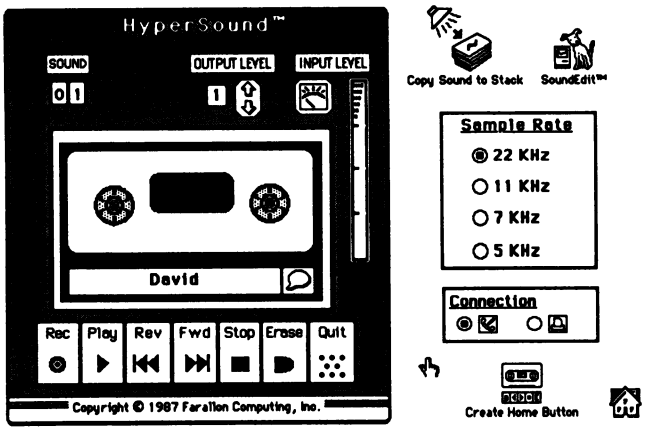
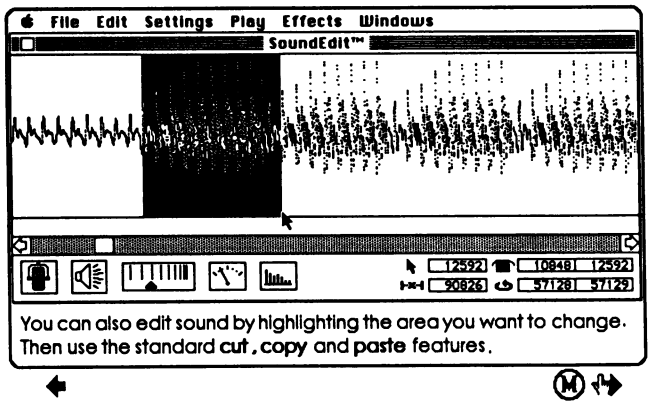


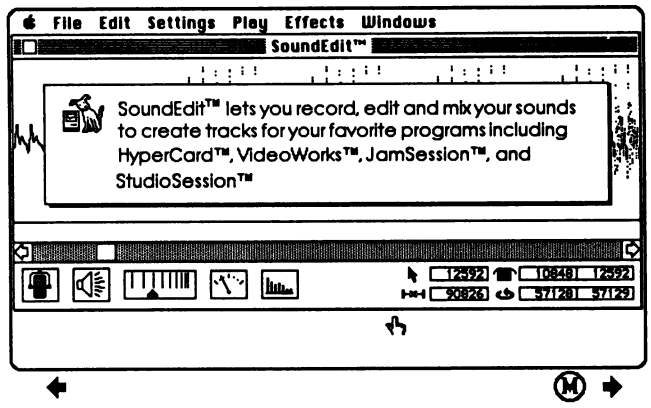
Fig. 7 HyperSoun d

SoundEdit allows you to edit sounds in a very natural manner(see figure 8), such as copy and paste if you have a color monitor you can take advantage of it by coloring particular bits of the sound waveform.



There are many different effects from echo, to a graphic equaliser filter. Naturally you can record with SoundEdit and you can store sounds in many forms as well as hypercard, including, VideoWorks, JamSession, and StudioSession.

Farallon pioneered the use of Hypercard to demonstrate products. They developed a stack called "The PhoneNet Demo Stack" which is an interactive stack wich uses sound to hilite the



demo. Developers are now seeing the merit in putting help stacks with their products, and also using Hypercard to Demo some of the product functionality without releasing a crippled demo version. This way you can guide your audience to the features without them having to blindly explore on their own.

Two other stacks are included in the MacRecorder package one is called MacRecorder which is a guide and includes some good sound examples.

The second is a stack called MacRecorder Applications and this shows examples of applications that can be made with sound. A good example is a face and as you click on parts of the face, up comes the name in English and French and you here the French word Spoken. Don't get your hopes up too high, nobody yet, has written a script that will allow you to speak into a MacRecorder in English and have the sound come out in French.

If you are planning on doing any serious Hypercard development I would recommend the MacRecorder package from Farallon it is one of the most professional packages I have seen and at under $200 it is great value.

Please send your ScriptTips, Stacks, HyperArticles, Comments, and Ideas to:

Applelink D0280 for fastest turnaround or
Fred Stauder
Ecofin Research and Consulting
Lavaterstrasse 45
Zurich 8002 Switzerland
or to the MacTutor office.

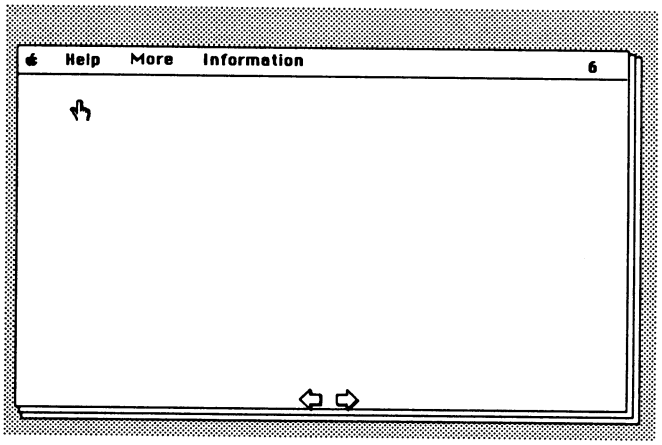
HyperChat™

PopUpMenus in HyperTalk

Joseph Bergin, Professor
Marist College, NY
MacTutor Vol. 4 No. 5

When Hypercard first appeared it wasn't possible to include user defined menus in your stacks. While waiting for my Hypercard developers kit to arrive from APDA I decided to experiment and to see what could be done without resorting to XFCN's and XCMD's. The result of my early trials is shown in Figure 1. It looks similar to a menu bar except that I placed it on the top of a "card" instead of the top of the window. Actually what you see is a collection of invisible labeled buttons drawn across the top of a card and not an actual menu bar.

In Smalltalk a popup menu is a menu selection device which appears (pops up) when you click the mouse button on certain screen areas. These areas do not need to be anywhere special, and they do not need to be labeled. You select an option by clicking (not releasing) on one of the choices. For example the main control window pops up when you click on the background outside of any window. Another difference between Apple's pull down menus and Smalltalk's popups is the fact that popups may stay on the desktop while you work until you actually dismiss them, usually by taking some menu choice. This is similar to Apple's new tear off menus that made their appearance in Hypercard.



When you click on one of these buttons a "Popup Menu" appears. If we click on the "Apple" we get Figure 2. This popup is actually a field. Clicking on the Apple only made it appear. As you can see it has four choices presented to the user. If the user clicks on one of these the corresponding action will be taken and the popup will disappear. If you don't take an action the popup will just sit there while you do other things. The popup will also disappear if you move the browser (cursor) into the window and then move it out again without clicking. This just "dismisses" the popup.

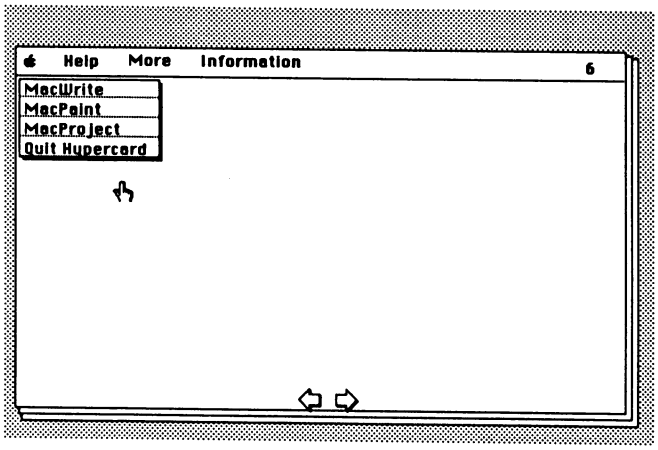


Fig. 2 The HyperCard PopUpMenu

The purpose of popup menus is to provide for those situations in which you want to provide a lot of options for your user but you don't want to clutter up the cards and/or backgrounds with a lot of buttons which add visual "noise" to the interface. The options may be clustered together in groups and each group may be provided with a single button to call up a popup with the associated choices. There are a number of other solutions to this problem but if used wisely popup menus can be a valuable addition to your Hypercard toolkit.

Actually, what I learned about HyperTalk along the way to creating these popups was as valuable as the popups themselves. In order to make the creation of popups easier I decided to automate the process. This made it unnecessary for me to remember all of the steps that were needed in creation of a popup and its associated button, field, and scripts. The solution was to create a command which creates new popups, but it does so by having a script that writes scripts.

In the Unix world a program which writes other programs is a fairly standard item (YACC and LEX for example) but they seem to be less used in other environments. They are an important development tool, however, and quite easy to do in Hypercard.

The necessary scripts

Before we get into the operation of the ScriptWriter we should look first at the scripts that are needed for the popups to operate correctly.

Suppose that we want to have a popup called "Apple" and we want it to look like the first example in this article. We will need a background button called "Apple" which, when clicked shows a background field called "Apple." The script to do this is simply

```
on mouseUp
     show bkgnd field Apple - at the clickLoc
     - remove the dashes above for a
     -nice effect
end mouseUp
```

which belongs in the script of the "apple" button.

Next we need the background field called by the same name, "Apple." It has a script to hide itself (target) and one to select and execute the required user choice, which here is done on mouseUp. It also has a special action command ("apple1", "apple2", etc.) for each visible line of the field. You may have more action commands than visible lines if you like but they won't be usable. You can also add new action commands at any time by editing the script, and increasing the number of visible lines in the field and all will be well. Note though that the lockText property of the field will have to be set to true so that Hypercard will be able to recognize a click in the field. The required scripts for this field are:

```
on mouseLeave
  hide the target
end mouseLeave

on mouseUp
  selectChoice
end mouseUp

on apple1
  open MacWrite
end apple1

on apple2
  open MacPaint
end apple2

on apple3
  open MacProject
end apple3

on apple4
  doMenu "Quit Hypercard"
end apple4
```

Recall that the second choice on the "Apple menu" was to open Mac Paint. This will happen when we execute the command apple2. The command selectChoice decides that the mouseup happened on line 2 of the popup and dispatched the command apple2.

The small scripts appleN can contain anything that you like. They can show or hide fields or buttons. They can paint. They can do menu commands or open files or stacks. They should be kept short, though, perhaps by defining new commands that they call.

After the field was created and the scripts put into it the lockText was set to false so that its contents could be edited. Then a line was entered into the field for each possible choice, indicating what the choice is. Then the lockText was set to true.

The script for selectChoice is either in the stack script or, perhaps, in the script of the Home Card. This command is called when the user mouses down in a popup. It first computes the line number on which the user clicked the mouse and then executes a command whose name is the same as the name of the field which took the click with the line number appended to it. Thus, if the user clicked on line 3 (selection 3) of a popup whose field was called Apple, this script would execute command apple3, which was shown above.

```
on selectChoice -used by popups
  put the short name of the target into theName
  get the rect of the target  -field clicked in
  put   item 2 of it into top -top of the rect
  get the textHeight of the  target
  put it into size -how big each line is
  put ((item 2 of the clickLoc) - top + size) ¬
     div size into lineNumber
  send ( theName&linenumber) to target
  - executes the required command
end selectChoice
```

How to create popups

Since there is so much to do and type for popups it would be nice to automate their creation. The following shows how. It assumes that you want your popup buttons and fields that define the popup to be part of the background, and that if you create a new card from the background you will want the popup to look and behave the same in the new card. None of these assumptions are required and you might experiment to discover the effect of doing it differently. That would require modifying the script for the command "newPopup" which is described next.

To create a new popup called Apple, with four options, in a stack that contains all of these scripts you just type

```
newPopup Apple,4
```

into the message window and the button and field will be created and the required scripts will be put into them. You will then need to resize and place the field in the card, type your menu options into it, lock its text, and edit the action commands to give the effect that you want. The action command skeletons

```
on Apple1

end Apple1
```

etc. will be created for you however.

The script that does all this work is shown below. It uses three other commands which are shown later. First, newPopup, which has two parameters, theName and theNumber, locks the screen to speed things up, says that we want to do this stuff to the background, and remembers the current tool so that it can be restored to the user at the end. Next it calls the first auxiliary command to create the button which will bring up the popup. Then the next command creates the popup itself, and the last command in the group adds a script segment to the background so that new Card operations will perform as suggested above. Finally the user's state is restored and the screen unlocked. [I have done some modifications to the script which will allow a

windoid type menu that can be draged around. The original script is intact; there are comments how to modify it. The modifications are in italics. -HyperEd]

```
on newPopup theName,theNumber
  — add next line for watch cursor HyperEd
  - set cursor to 4
  —

  set lockScreen to true
  set editBkgnd to true
  put the tool into oldTool


  doNewButton theName

  — add this line HyperEd
  - doNewBtnClose theName
  —

  doNewField theName, theNumber

  set editBkgnd to false

  fixBkgnd theName

  choose oldTool
  set lockscreen to false

  — Return to normal cursor HyperEd
  - set cursor to 1
  —

end newPopup
```

[The next section puts in the button for the close box. HyperEd]

```
-on doNewBtnClose theName    - This whole procedure HyperEd
- set editBkgnd to true
doMenu "New Button"
set name of bkgnd button "New Button" to "close"
set style of bkgnd button close to checkbox
set autoHilite of bkgnd button close to true
set showname of bkgnd button close to false
set rect of bkgnd button close to 206,73,226,90
put "on mouseup" into newScript
put "hide bkgnd field" && theName into line 2 of newScript
put "hide bkgnd btn close" into line 3 of newScript
put "set hilite of me to false" into line 4 of newScript
put "set hilite of bkgnd btn" && theName && "to false" ¬
into line 5 of newScript
put "end mouseUp" into line 6 of newScript
set script of bkgnd button close to newScript
-end doNewBtnClose
```

Next we need to examine the three auxiliary commands that newPopup used. The doNewButton command is the easiest and it illustrates the technique of scripts which write scripts. The first line of it just creates a new button. It doesn't say background button, but it will be because of the fact that it is called after setting edit bkgnd to true. The next five commands just set some properties of the new button to something convenient. Note that the New Button menu command should set the new button's name to New Button. If it doesn't, you will need a script called newButton which just sets the name of "the target" to "New Button". We will need to do a similar thing for new fields also.

```
on doNewButton theName
  - set editBkgnd to true
  doMenu "New Button"
  set name of bkgnd button "New Button" to theName

  — change next line to shadow HyperEd
  set style of bkgnd button theName to transparent
  —

  — change next line to false HyperEd
  set autoHilite of bkgnd button theName to true
  —

  set showname of bkgnd button theName to true
  set rect of bkgnd button theName to 206,70,306,92
  put "on mouseup" into newScript
  put "show bkgnd field" && theName into line 2 of newScript

  — add these three lines HyperEd
  - put "show bkgnd btn close" into line 3 of newScript
  - put "set hilite of bkgnd btn" && theName && "to true"¬
  - into line 4 of newScript
  —

  put "end mouseUp" into line 5 of newScript

  -- This Section added take away "-" HyperEd
  - put "on mousestilldown" into line 8 of newScript
  - put "set lockscreen to true" into line 9 of newScript
  - put "if the visible of bkgnd field" && theName && "is true
then" ¬
  - into line 10 of newScript
  - put "set the loc of me to the mouseloc" into line 11 of
newScript
  - put "set the loc of bkgnd btn close to  the mouseh -36,
the mousev" ¬
  - into line 12 of newScript
  - put "set the loc of bkgnd field" && theName && "to the
mouseh, the mousev+93"¬
  - into line 13 of newScript
  - put "end if" into line 14 of newScript
  - put "set lockscreen to false" into line 15 of newScript
  - put "end mousestilldown" into line 16 of newScript
  — End Section HyperEd

  set script of bkgnd button theName to newScript

  — added HyperEd
  - set hilite of bkgnd btn theName to true
  —

end doNewButton
```

[These sections ad the script neccessary to move the Drag-Menu. HyperEd]

The next four commands write a script into the new button. Suppose that the user had originally typed newPopup Apple, 4 into the message window. Then theName is Apple, and theNumber will be 4. After we execute the first of these script writing commands the local variable newScript has a value:

newScript=
    on mouseUp

Then after the second we will have:

newScript=
    on mouseUp

```
show bkgnd field Apple
```

Note that && concatenates strings but leaves appropriate spaces between words. We could easily modify this to show the bkgnd field at the location of the users click. we would just insert

&& "at the clickLoc"

after the word theName in the second put command. After the next command our script is:

newScript=

```
on mouseUp
   show bkgnd field Apple
end mouseUp
```

The fourth command copies this to the script of our new button, whose name by now is "Apple" which is the value stored in theName.

The next auxiliary script, doNewField, is a bit more elaborate but not much harder. It needs to know both the name and the number of action commands as these will be written into the script of the field. Again we simply create the field with the doMenu command and then set some of its properties to useful values. The next seven commands then write the mouseUp command for the field. It is left in local variable newScript, as before. After further augmentation we will put it into the script of the field. That will come at the end, just before we end this method.

```
on doNewField theName, theNumber
  -set editBkgnd to true
  doMenu "New Field"
  set name of bkgnd field "New Field" to theName
  set showlines of bkgnd field theName to true

  set style of bkgnd field theName to shadow


  set textFont of bkgnd field theName to Chicago
  set textSize of bkgnd field theName to 12

  -- This line added HyperEd
  - set the rect of bkgnd field theName to 206,93,306,256
  --

  -- Remove the next four lines HyperEd
  put "on mouseleave" into newScript              -
  put "hide the target" into line 2 of newScript  -
  put "end mouseleave" into line 3 of newScript   -
  put " " into line 4 of newScript                -
  --

  put "on mouseUp" into line 5 of newScript
  put "selectChoice" into line 6 of newScript
  put "end mouseUp" into line 7 of newScript

  repeat with x = 1 to theNumber
    put " " into line (8 + (x-1) * 4) of newscript
    put "on" && theName&x into  line (9 + (x-1) * 4) of
newScript
    put " " into line  (10 + (x-1) * 4) of newScript
   put "end" && theName&x into line  (11 + (x-1) * 4) of
newScript
  end repeat

  set script of bkgnd field theName to newScript
end doNewField
```

The only new feature here is the repeat command which writes the action command skeletons onto the end of newScript. Note that at this point newScript has seven lines. Each time that the loop body is executed we put four more lines into the script. The first line is blank. The next will say, for our example,

on Apple1

or "Apple2..."

depending on the value of x which will vary between 1 and theNumber. This numeric value is appended to theName. The third line is blank again and the fourth gives the end line for the script. Finally we stuff newScript into the field script and quit.

Both of the above scripts were simple in that we were starting fresh with a new object and were writing complete scripts. The last script in this group writes a script segment into the newCard method of the background script for the current background. It assumes that if the background has such a method then it contains a line somewhere with nothing on it but the comment:

—here—

The modification will be done just above this line. If your background script doesn't have a newCard method at all then all is well and this script will create one with the required comment line.

```
on fixBkgnd theName
  get script of bkgnd
  put it into bgscript
  put the number of lines of it into numLines
  put "no" into found
  repeat with x = 1 to numLines
    get line x of bgscript
    if  it  contains  "-here-"  then
    put "yes" into found
    put x into where
    exit repeat
    - got it
    end if
    end repeat
    get the number of this background
    put it into bknum
    if found = "no" then
    put return & "on newCard " && return   ¬
     after last line of bgscript
    put return after last line of bgscript
    put "set lockScreen to true" & return ¬
     after last line of bgscript
    put "push card" & return after last line ¬
     of bgscript
    put "go to recent card" & return after ¬
     last line  of bgscript
    put "get bkgnd field" && theName & return ¬
     after last line of bgscript
    put "pop card" & return after last line of ¬
     bgscript
    put "put it into bkgnd field" &&  theName & ¬
     return after last line of bgscript
    put "-here-" & return after last line ¬
     of bgscript
    put "set lockScreen to false" & return ¬
     after last line of bgscript
    put "end newCard" & return ¬
     after last line of bgscript
  else
     put "push card" & return before ¬
     line where of bgscript
```


© The Definitive MacTutor, Vol. 4

```
    put "go to recent card" & return before ¬
     line where +1 of bgscript
    put "get bkgnd field" && theName & return ¬
     before line where +2 of bgscript
    put "pop card" & return before ¬
     line where +3 of bgscript
    put "put it into bkgnd field" && theName ¬
     & return before line where +4  of bgscript
  end if
  set script of  background bknum to bgscript
end fixBkgnd
```

The first few lines of this script search for the magic comment —here— in the script. If it is not found a newCard method with all the required parts is written into the background script copy which is now stored in the local variable bgscript. Not much new here, except that we are appending to an existing script which explains why we include the "returns" at the beginning of various put commands. If the comment was found on line "where" then the required script is inserted just before it. The key part of the script that we write into the variable bgscript, for later stuffing into the background script itself, is just a way to copy the contents of the popup field from the original card into the field for the new card. This will make the popup menu have the same choices as the original card had. We must first go back to the original card, copy the field contents, go back to the new card and then put the contents into the old bkgnd field but from the new card. This requires writing a sequence push-go-get-pop-put into the background script each time we create a new popup, so that the field will get copied each time we create a new card. Simple. Except that the line counting gets a little tricky in modifying the script as we are repeatedly modifying it and changing linecounts.

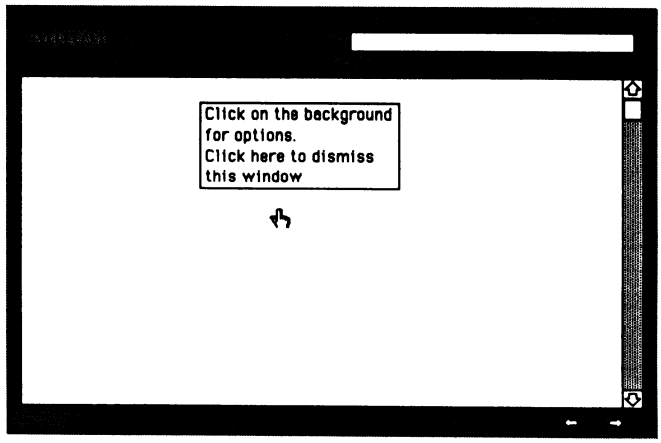
Anyway, in spite of the complication of the scripts, the use is quite simple. Just type newPopup "someName", "someNumber" into the message window and a new popup will appear in your stack as if by magic.

You can also refine your popups nicely by modifying the scripts slightly. For example, you might want to have a popup which has different choices in it at different times, or shows the choices in different ways. Here is a script (from the Home Card or the stack script) which puts a check mark at the beginning of a line of a popup. Just remember that lines count between actual "returns" in Hypercard. Auto wrap lines don't count. Call this from your action command scripts

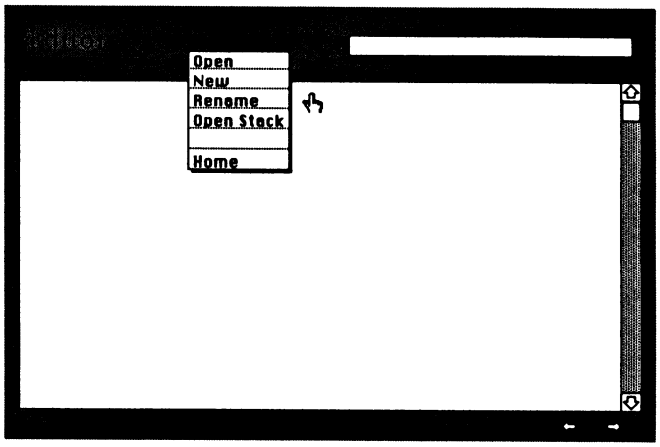
```
on wantCheck  fieldName, lineNumber
  - toggles a checkmark at the beginning of a
  - line of a bkgnd field
  get line linenumber of bkgnd field fieldName
  if char 1 of it <> "✓" then
    Put "✓" before it
  else
    delete first char of it
  end if
  put it into line linenumber of L
     bkgnd field fieldName
end wantCheck
```

Interface guidelines

Popup menus that look like ordinary menus but which behave differently will not be admired by your users. They will be expecting "Press-drag-release" to choose, and if they need to "click-move-click" instead it will only add to their frustration. The solution is to make them look different. That is to say, the buttons which summon the popups should not be arranged in the form of the first card shown in this article, all in a row across the top of the window. Words down the left might be OK. Icons instead of words would probably be acceptable. One of the best methods is to cover part of your background with a large invisible button which has an invisible name as well. Stack instructions (perhaps summoned by a button press) could explain that options may be chosen by first "clicking near the upper left of the card." If nothing else is in front of your invisible button (or even if it is painted over by card graphics) your popup will be appear. One of the stacks that I recently built is a multi window editor that is controlled by popups. The next figure is the screen you see when the stack is first opened.

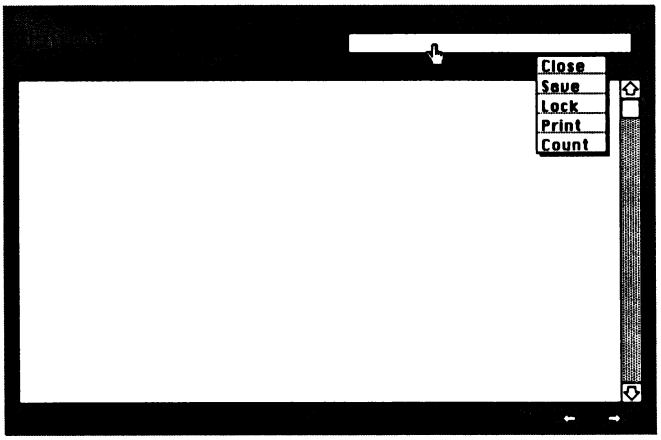


This background of this card is covered by a large button. Any clicking on an area not covered by some other item will cause the popup shown to appear. In the next figure the user has dismissed the instructions and has just clicked on the background. The popup also draws itself at a fixed location, but it could be at the point of the mouse click (in HyperScript this is the ClickLoc).



This stack has another feature that you may choose to implement. One of the popups in it is controlled by clicking on the "filename" field in the upper right, rather than on a button.

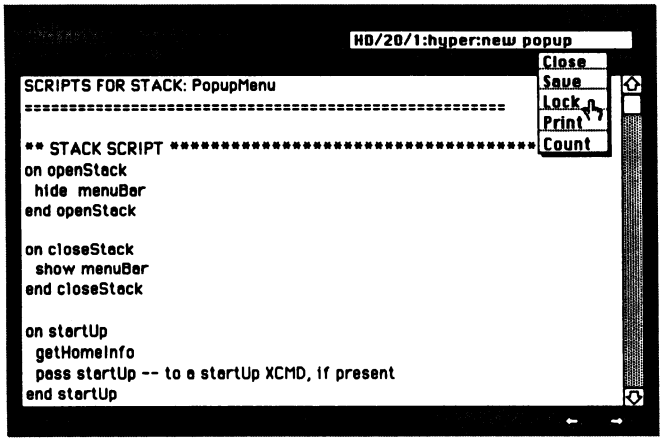
This field normally contains the name of the file being edited and islocked text, filled in by the scripts.



In operation a card will show the text to be edited. By creating new cards from this background you may have a multi window editor for text files. It is also useful as a note pad.

Be careful with popups that don't disappear on mouseLeave. They should have at least one "Safe" option among the choices so that the user can make them them just disappear.

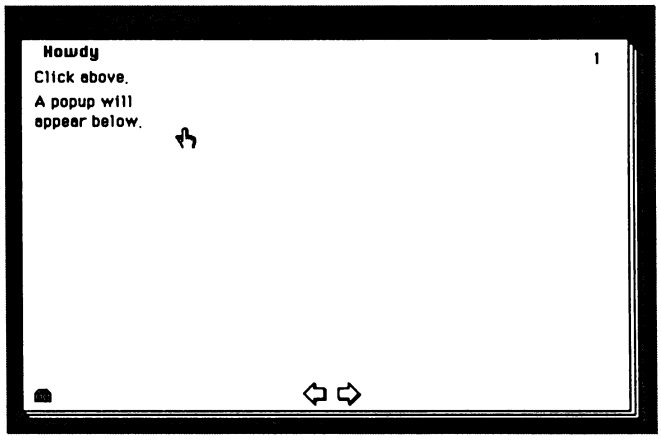
You might want to experiment with popups which disappear on mouseLeave as well as on down or up. A combination of Up and Leave is probably best. Note that if the popup disappears at mouseUp you can mouse down and then drag off the popup, and your initial mouseDown will be ignored. This lets your user avoid a choice. This is not possible with a choice at mouseDown.



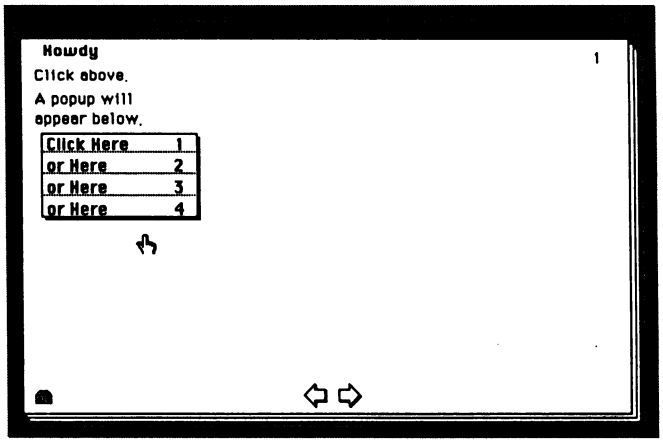
What you get on this month's disk

This article is packaged with two stacks. The first is a practice stack to help you learn about popups. It also contains all of the necessary scripts, especially newPopup. The other stack is the simple editor described above that uses popups for its commands. This stack has only one card in it, but by executing "new Card" from the Edit menu you will build yourself a multi window editor for text files. It is also a useful textfile viewer. For this stack to work correctly you will need to paste the newPopup and selectChoice scripts, as well as the auxiliary scripts, either into the Editor stack or into your Home Stack. The latter is preferred so that you can have popups available in all your stacks.

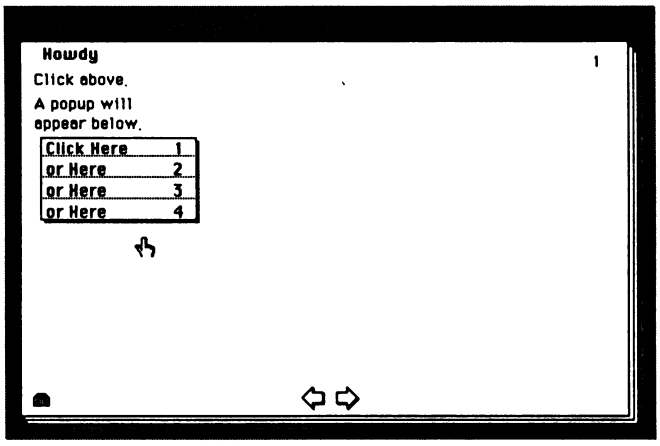
The following figures are screen shots from the popup stack which contains all of these scripts and a few more goodies.

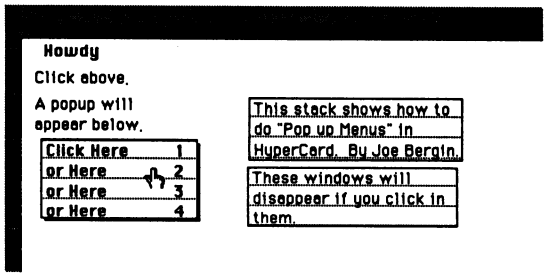


The popup is called Howdy, and its purpose is to call up windows that teach you about popups and how to create them. The first figure in this article is a card from this stack.

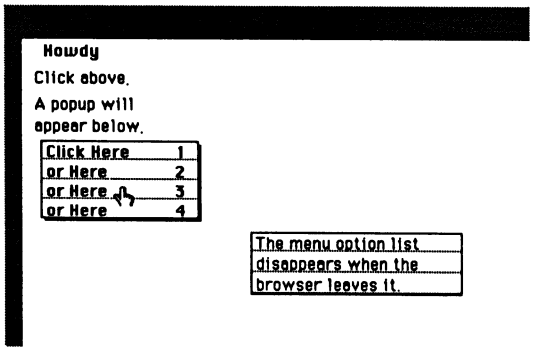


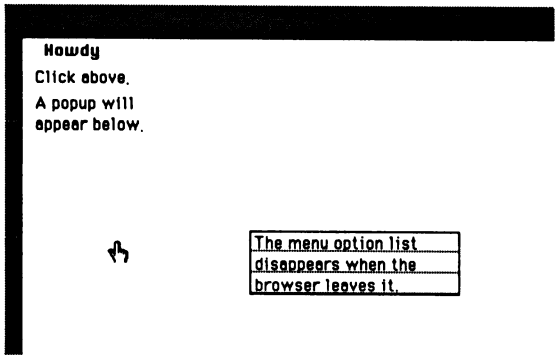
When Howdy is clicked it brings up a popup with four choices. Each of these will bring up another window which has information about popups in it.





Line 1 and 2 have been clicked. The popup won't scoot until the browser leaves it. Thus you may make several choices on the same menu. Did you ever want to do that in standard Mac menus?

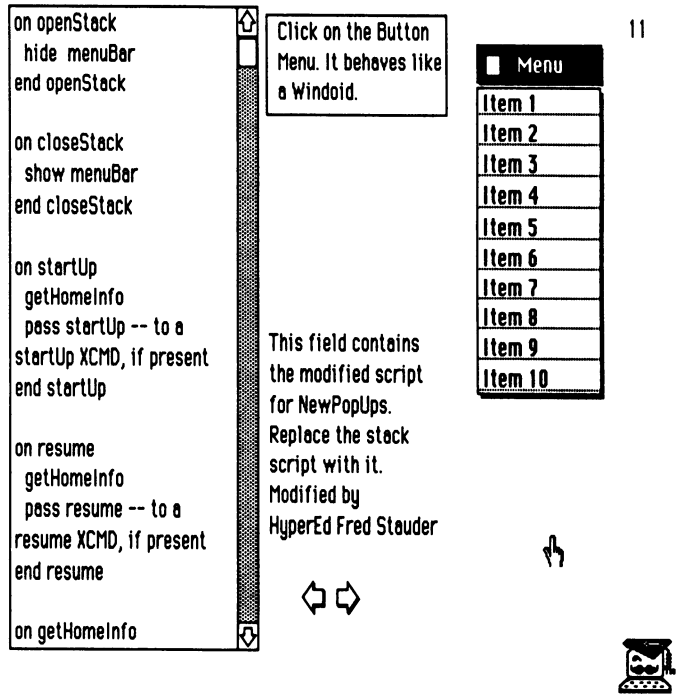




The user has clicked in the two windows from before. The popup disappeared, but it was called back and the user clicked on the third line.

The user moved the browser from the popup and it went away. Clicking in the window showing will dispel it as well. Once you get used to fields as menus it is relatively simple to create hierarchical menus. What you need to do is have a script for one of your action commands call up another popup. It is easy to overdo this however, and your users won't be too happy if every popup calls others. One of the most frustrating features of older computers (pre Macintosh) was having to wade through menu levels when you already knew what you wanted. Ah well, it's nice that times have changed. Happy Computing.

[Figure 12 Shows the Windoid type DragMenu. The method of scripts making objects is very powerful. As you can see from my additions a different type of menu can be produced. Experiment and see what you can come up with. HyperEd]



HyperChat™

New HyperCard 1.2 Features

Fred Stauder
HyperChatEditor
Zurich, Switzerland
MacTutor Vol. 4 No. 6

They are listening!!

Last month I told you to be patient with your wish lists. Well the good folks at Apple have answered some of your wishes. Hypercard 1.2 will have announced by the time you read this issue. The main emphasis of this issue will be the new Hypercard features in 1.2, also Don Koscheka joins us in bringing you XCMD Cookbook.

New Features in Hypercard 1.2

First of all Hypercard 1.2 is not the final version of Hypercard, it is however an indication from Apple that they are committed to the evolution of the Hypercard paradigm. If you expected all the bells and whistles in this release you will be dissapointed. However Hypercard 1.2 does address lots of important points.

HyperCard stacks can be write-protected

This is a very important feature that allows you to get some performance improvements where they are criticle, such as a terminal stack. You can, for example, open it locked and have a save button that unlocks the stack and saves the field to another card.

You can recognize a write-protected stack by a padlock icon appearing to the right of the menu bar. When a stack is write-protected, you can't type into any fields (even if they're not locked) and you can't use certain menu items (they're dimmed).

You might be able to make temporary changes to a card if the userModify property is set to true. But as soon as you leave the card, the changes will disappear.

A stack is considered write-protected under the following circumstances:

- The stack is locked
- Can't Modify Stack is checked in the stack's Protect Stack dialog box.
- The cantModify property is set to TRUE.

When the stack is locked, HyperCard automatically puts a check into the Can't Modify Stack option; you can't uncheck it.

Stacks on File Servers and CD-ROM

HyperCard 1.2 allows more than one user to browse stacks on file servers. Any number of users have access to a stack when the stack is locked (but no one can make changes to it); only a single user has access to a stack on a file server when the stack is unlocked.

A stack is considered locked if any one of the following conditions are true:

- The stack is on a CD-ROM.
- The stack is on a file server in a folder whose access privileges are set to Read Only.
- The Locked box is checked in the stack's Get Info box in the Finder.
- The stack is on a locked 3.5-inch disk.

Note: A locked stack is considered write-protected (see next card), but checking Can't Modify Stack in a stack's Protect Stack dialog box does not lock it (even though it does write-protect the stack). Only locked stacks support browsing by more than one user.

Auto Tabbing in Fields

The Auto Tab option lets the Return key move the insertion point to the next field on the card when in the last line of a non-scrolling field. Auto Tab is a property of card and background fields. When Auto Tab is checked in the Field Info dialog box of a non-scrolling field, pressing Return with the insertion point in the last line of that field moves the insertion point to the next field on that card. Normal tabbing order is followed. Remember field order can be set by bringing the field closer or further, the order can be checked by the field number in the field info box.

Setting up form type cards is now much easier where entry can be done entierly via the keyboard.

Choosing Button and Field Tools

The shortcut for the browse tool is command-tab. Two new tool choices have been added. To choose the Button tool command-tab-tab, and to choose the field tool command-tab-tab-tab. See —ScriptTips AutoScriptEdit-II for an alternative.

Line Spacing Shortcuts

In earlier versions of HyperCard you could press Command-Option > and Command-Option < to increase or decrease the space between lines of text (the textHeight property of a field or of paint text).

In version 1.2, you must add the Shift key to the formula: The main reason for this is to be compatible with international keyboards.

First, select the field whose line spacing you want to change. Then press

Command-Shift-Option >	to increase the spacing between lines
Command-Shift-Option <	to decrease the spacing between lines

Each shortcut increments or decrements the line spacing by 1 (for example, from 16 to 17 or from 17 to 16).

New Hypertalk Commands

Find Command

Find Whole and Find String are two new options for the Find

© The Definitive MacTutor, Vol. 4

Command. Find Whole (or Shift-Command-F) lets you search for a specific word or phrase (including spaces). It searches for characters at the beginnings of words. Find String lets you search for a specific string of characters (including spaces), but it ignores word boundaries.

Hide Picture Command

Hide Picture hides the picture of a specified card or background. Hide Card Picture and Hide Background Picture hide the picture of the current card or background. This is very useful as an extra graphic layer.

Lock and Unlock Screen Command

Lock Screen has the same effect as setting the property lockScreen to true: it prevents HyperCard from updating the screen. Unlock Screen has the same effect as setting the property lockScreen to false: it allows HyperCard to update the screen. Unlock optionally allows a single visual effect.

Select Command

The Select command lets you select buttons, fields, or text in a field. Select Text and Select with a chunk expression let you (a) highlight text in a field or (b) position the insertion point in a field.

Show Picture Command

Show Picture shows the picture of a specified card or background. Show Card Picture or Show Background Picture shows the graphics of the current card or background.

clickH and clickV Function

The clickH returns an integer equal to the number of horizontal pixels from the left side of the card window to the place the mouse was last clicked

(item 1 of the clickLoc). The clickV returns an integer equal to the number of vertical pixels from the top of the card window to the place the mouse was last clicked (item 2 of the clickLoc).

foundChunk Function

The foundChunk returns a chunk expression that indicates where the most recent Find command located its argument. Its result has the form:

char <number> to <number> of <card | bkgnd> field <number>. If nothing was found, it returns the empty string.

foundField Function

The foundField returns a field expression for the field in which the most recent Find command located its argument. Its result has the form: <card | bkgnd> field <number>. If nothing was found, it returns the empty string.

foundLine Function

The foundLine returns a line expression for the line of a field where the most recent Find command located its argument. Its result has the form: line <number> of <card | bkgnd> field <number>. If nothing was found, it returns the empty string.

foundText Function

The foundText returns the characters enclosed in the box after the most recent Find command has located its argument. If nothing was found, it returns the empty string.

Number Function

The "number of card of <background>" is a new option for the function Number. It returns the number of cards in the specified background.

selectedChunk Function

The selectedChunk returns a chunk expression that indicates the range of characters currently highlighted. If nothing is selected, it returns the empty string.

selectedField Function

The selectedField returns a field expression for the range of characters that is currently highlighted. If nothing is selected, it returns the empty string.

selectedLine Function

The selectedLine returns a line expression for the range of characters that is currently highlighted. If nothing is selected, it returns the empty string.

selectedText Function

This function returns the text that is currently highlighted. It yields the same result as "the selection." Note, however, that "the selection" is a container (for example, you can put text into it); the selectedText is only a function. If nothing is selected, the selectedText returns the empty string.

autoTab Property

When autoTab is set to true, pressing the Return key with the insertion point in the last visible line of a field advances the insertion point to the next field on the card. Setting the autoTab property is the same as checking Auto Tab in the Field Info dialog box of a specific field.

cantDelete Property

The cantDelete property controls whether a user can delete the specified object. This property checks or unchecks the appropriate option on the Info dialog box of the object in question.

cantModify Property

The cantModify property lets you control whether a stack can be changed in any way. This property sets both the Can't Modify Stack option and the Can't Delete Stack option in the Protect Stack dialog box. When cantModify is true, the padlock symbol appears on the menu bar.

Cursor Property

The Cursor property has four new synonyms and four new options. Setting the cursor to iBeam, plus, cross, or watch is the same as setting the cursor to 1, 2, 3, or 4. The others are new cursors. (Note: "None" removes the cursor from the screen. This is very useful for animation you don't get the flickering browse tool which was very distracting. "Busy" brings up the beach ball. Each time you set it, the ball rotates an eighth of a turn.)

showPict Property

The showPict property makes the background or the card picture visible (when set to true) or invisible (when set to false).

userModify Property

userModify is a global property that allows the user to type in fields and use the Paint tools in a stack that has been write-protected. Changes made by the user (or a script) are discarded upon leaving the card. This property is set to false when the user changes stacks or quits HyperCard. userModify has no effect on an unlocked stack.

New Synonyms in Hypertalk

abbr, abbrev, abbreviated

bg [new], bkgnd, background

bgs [new], bkgnds, backgrounds
btn [new], button
btns [new], buttons
cd [new], card
cds [new], cards
char, character
chars, characters
fld [new], field
flds [new], fields
grey [new], gray
loc, location
mid, middle
msg, message
pict [new], picture
poly, polygon
prev, previous
rect, rectangle
reg, regular

Script Peeking Shortcuts

HyperCard 1.2 has several shortcuts for peeking at scripts. The user level must be set to Scripting (userLevel = 5) to peek at scripts.

While using the Browse tool:

Command-Option: Peek at all visible buttons; click one to see its script.

Shift-Command-Option: Peek at all visible buttons and fields; click one to see its script.

While using the Field tool:

Command-Option: Peek at all visible and invisible fields; click a visible field to see its script.

While using the Button tool:

Command-Option: Peek at all visible and invisible buttons; click a visible button to see its script.

While using any tool:

Command-Option-C: Peek at the script of the current card

Command-Option-B: Peek at the script of the current background

Command-Option-S: Peek at the script of the current stack.

To put any script away quickly once you are in the script editor, press Command-Option again and either (a) click once or (b) press any key.

I hope we can all remember these key combinations; does it remind you of IBM function key nightmares? How about having them in hieracial menus for the times we forget and the not so power users and developers that would like to use them. If I remember my human interface guidelines correctly command keys should be a supplement to menus.

The New "Me" and "Target"

put the target — still puts the name of the object that last received a message

put target into myVar — puts contents of a field into a variable

put "Fred" into target — puts "Fred" into the field

Note: "put value of the target" is equivalent to "put target."

The above changes affect only operations on fields, not operations on buttons, cards, backgrounds, or stacks.

returnInField System Message

If the returnInField message is not intercepted by a handler, HyperCard checks to see if the autoTab property of the current field is true. If it is and the insertion point is on the last line of the field, HyperCard sends the tabKey message to the current field. Then if the tabKey message is not intercepted by a handler, the cursor moves to the next field on the card.

enterInField System Message

If the enterInField message is not intercepted by a handler and the contents of the current field have changed, HyperCard sends the closeField message to the current field.

Rects

You can now access parts of the rectangle of a specified object and test whether a point is contained within a rectangular region.

In the following descriptions, the symbol <rect> means any expression that yields a rect. A rect consists of four integers, separated by commas, that specify the top-left corner and bottom-right corner of a rectangle as offset in pixels from the top-left corner of the card window. The order of the integers in a rect is Left, Top, Right, Bottom.

For example, the following are examples of expressions that yield a rect:

```
rect of card field 1
rect of first btn
rect of tool window
rect of pattern window
rect of msg box
rect of card window
```

- There is a new function, the screenRect, that returns the rect of the entire screen in which the HyperCard window resides. Since it is a function, you cannot use it with Set.

```
get the screenRect
```

- Using <rect>, the following new properties can be defined:

```
left of <rect>
is equal to item 1 of <rect>

top of <rect>
is equal to item 2 of <rect>

right of <rect>
is equal to item 3 of <rect>

bottom of <rect>
is equal to item 4 of <rect>

topLeft of <rect>
is equal to item 1, item 2 of <rect> (or the top-left corner point)

bottomRight of <rect> or botRight of <rect>
is equal to item 3, item 4 of rect (or the bottom-right corner point)
```

width of <rect>
is equal to (item 3 of <rect> - item 1 of <rect>)

height of <rect>
is equal to (item 4 of <rect> - item 2 of <rect>)

• You can use Set with left, top, right, bottom, topLeft, and bottomRight to move the specified object. For example, "set left of background button 3 to 25." Note: You cannot Set the screenRect. It is a pity you could do some interesting things with it.

• You can use Set with width and height to resize buttons and fields.For example, "set width of background button 3 to 25." Note: You cannot resize the message box, the tool window, the pattern window, the card window, or the screenRect. When changing the width or height, HyperCard maintains the location or loc (center coordinate) of the object, expanding or shrinking it on both sides evenly. If either property is changed by an odd amount, then HyperCard adds the odd pixel either on the right (with width) or on the bottom (with height).

• HyperCard Version 1.2 has a new operator, Within, that allows you to ask if a point is contained in the region defined by a rect. It returns a boolean value (true or false) and is used as follows:

<point> is within <rect>
<point> is not within <rect>

For example:

"12,34" is within "45,45,100,100"
the clickLoc is within the rect of button 3
the mouseLoc is not within the rect of me

Consistent with the Macintosh's rules about rects, the top left point of a rect is within the rect; the bottom right point of a rect is not within the rect.

As you can see there are many new features in Hypercard 1.2 take the time to explore them fully, there is a lot more power in these changes than is obvious at first. Next month I will show you some example stacks that take advantage of the new features. Some of this information was obtained from the vesion 1.2 release notes from Apple Computer.

ScriptTips

The response to AutoScriptEdit was so intense that I decided I had to expand it and make an even more powerful version. AutoScriptEdit-II will get the script of an unlocked field as well as locked fields. I thought what are some of the frustrating things that can be improved upon. The answer I came up with was locking and unlocking fields. Now if you hold down the enter key and mouse button the field will toggle between locked and unlocked. I am using mouseStillDown because it is used much less than mouseDown. Another feature that I have added, to demonstrate how one key can serve two purposes, is showing the tool window at the mouseLoc using option-click. This is much more useful than the standard option-tab because it pops up where you want it. I have used AutoScriptEdit to demonstrate the power of gestural commands, i.e. holding the mouse down and moving off the top of the card to get the stack script.

```
— AutoScriptEdit-II by Fred Stauder  May 1988 —

- Put this Script into your Home Stack
- To use keep mouse btn down over Button or Field andmove -
mouse up
- To edit Bkgnd Script keep mouse btn down move mouse to -left
of card
- Card Script - move right
- Stack Script - move up
- Now works with unlocked fields
- To lock and unlock fields option click in them
- To show tools at cursor hold option click on non-  field
areas

on mouseStillDown
  if the optionkey is down then
    if the target contains "field" then set locktext of target
to false
    else show tool window at the mouseloc
  end if
  put char 6 of the target into C6
  if C6 is quote then AutoScriptEdit
  if C6 is "i" then AutoScriptEdit
  if C6 is quote then exit mousestilldown
  if C6 is "i" then exit mousestilldown
  if the mouseV <item 2 of rect of target then edit script of
target
  pass mouseStillDown
end mouseStillDown

on AutoScriptEdit
  if the mouseH < 2   then edit script of this bkgnd
  if the mouseH > 510 then edit script of this card
  if the mouseV < 2   then edit script of this stack
end AutoScriptEdit

on openfield
  if the mouseV <item 2 of rect of target then edit script of
target
  if the optionkey is down then set locktext of target to true
  pass openfield
end openfield

Non-Comercial use only - All Rights Reserved
Fred Stauder, Hypercard Editor, (HyperChat) MacTutor
```

SpeedTips

To get more performence out of your stacks especially with graphics you can "Preload" cards by setting lockscreen and going to those cards. This is particularly important if you are doing animations.

Another speed tip is to use the link button to get the id of a card rather than a name, it is a good idea to put the name in comments so that people reading your scripts have a better idea what is going on.

The last SpeedTip for this month is put commonly used information in globals rather than having to go to a card to get the info from a field. Send your Articles, ScriptTips, etc. to:

Fred Stauder
Ecofin Reseach and Consulting
Lavaterstrasse 45
Zurich 8002
Switzerland
Applelink D0280 or
To the MacTutor Office, PO Box 400, Placentia, CA 92670

HyperChat™
XCMD CookBook

Donald Koscheka
Apple Computer, Inc.
MacTutor Vol. 4 No. 6

[Donald Koscheka is a Software Engineer employed by Apple Computer Inc. He has written some very important development XCMDs which you will hear about soon. -HyperEd]

Introduction to XCMD's

If you've used Hypercard and its programming language HyperTalk for any length of time, you've no doubt discovered some of the things that you can't do with Hypercard. Perhaps you're writing a forms generator in HyperTalk and you want to be able to print customized reports. The designers of Hypercard knew that their product would have to be a lot of things to a lot of people and it would be nearly impossible to provide every possible feature in the language. They did, however, provide us with the capability of customizing HyperCard and HyperTalk directly. To paraphrase Abraham Lincoln, "You can fool some of the people all of the time, or all of the people some of the time, but to fool all of the people all of the time, write an XCMD".

As a programming language, HyperTalk is highly extensible; you can add your own commands and functions to the language very easily. These extensions are known as eXternal CoMmanDs (XCMDs) and eXternal FunCtioNs (XFCNs). The capital letters, ie. case, are significant as I'll discuss later. XCMDs and XFCNs are identical at the coding level, so I'll refer to both as XCMDs. The determination of whether a command will be an XCMD or an XFCN is made at link-time.

Creating an XCMD is a straightforward process. First, you write the XCMD using the language of your choice (I'll show examples in both "C" and Pascal). Next, you compile and link the XCMD as a resource and then you add it to the resource fork of the stack that you want to call it from, the home stack or even Hypercard itself.

Where you put the XCMD is significant. When Hypercard encounters a command that it does not recognize, it first checks the current object (button or field) to see if it contains any handlers for that command. If no handler is found, the search continues in the following order: card, background, stack, home stack, HyperCard. If after looking in all the places that it could expect to find a handler no handler turns up, Hypercard then searches its own resource fork for any external commands that can handle the command. It identifies the appropriate command by name. Note what this hiearchy implies. If you want your XCMD to be globally visible to all stacks, put the XCMD in HyperCard's resource fork or Home Card. There is a tradeoff, though. If you want the stack that uses the XCMD to be "portable", you'll need to put the XCMD in that stack. There is no conflict of interest here, you can put the XCMD in both places. *[Use Rescopy , via scripts, to allow the user the option of installing it into the Home Stack. -HyperEd]* Hypercard stops searching as soon as it finds a copy of the command. If it finds a copy in your stack, it'll stop the search there. If you call the XCMD from another stack that does not contain the XCMD, then it'll go all the way back to the Home Stack or Hypercard to search for the command.

Now that you have an idea of where to place the XCMD, let's take a look at how they are created. All XCMDs and XFCNs interface to Hypercard in a standard and straightforward way. When an XCMD is invoked, Hypercard will pass it a parameter block that contains, among other things, the number of parameters and handles to the parameters. For argument's sake, let's see how the following XCMD would be called:

```
Pizza "Cheese", "Pepperoni", "Anchovies"
```

When this XCMD is called, Hypercard will pass a record, referred to as the *Parameter Block*, to the XCMD. The first argument in the parameter block will be the number of parameters passed; in this case three, one for each item in the argument list. Each parameter is passed as a handle to a zero-terminated text string. Parameter 1 will be a handle to the string "Cheese\0", where the the '\0' means the character whose ASCII value is 0. "C" programmers will recognize this format as a standard string definition in "C". Pascal programmers often refer to this format as a *C-String*. The actual structure of a parameter block (in Pascal) is:

```
TYPE

  XCmdPtr = ^XCmdBlock;
  XCmdBlock =
    RECORD
      paramCount  : INTEGER;
      params      : ARRAY[1..16] OF Handle;
      returnValue : Handle;
      passFlag    : BOOLEAN;

      entryPoint  :  ProcPtr;  ( to call back to HyperCard )
      request     :  INTEGER;
      result      :  INTEGER;
      inArgs      :  ARRAY[1..8] OF LongInt;
      outArgs     :  ARRAY[1..4] OF LongInt;
    END;
```

Let's examine this record in more detail. First, we define a pointer to the record, called an XCmdPtr. Hypercard passes a pointer to the record so you'll need to be comfortable with

© The Definitive MacTutor, Vol. 4

pointers to work with XCMDs.

The first field in the record, *paramCount* , is a count of the number of parameters that have been passed to the XCMD. This is a number between 1 and 16, which is the maximum size of the parameter array, *params*, in field 2. Params is an array of handles which implies that the fundamental element in the array is a signed byte.

Field 3 in the record, *returnValue*, is a handle to the data that you want to return to Hypercard on completion of the XCMD. You create the handle and pass it back to hypercard with the assignment:

```
paramPtr^.returnValue := Handle_You_Created;
```

Herein lies the most important difference between XCMDs and XFCNs. ReturnValue os places in the global container, *result*, when returning from an XCMD and by value when returning from an XFCN. The following illustrates the mechanics:

```
XCMD: Pizza "cheese", "pepperoni", "anchovies"
    Put the result

XFCN: Put Pizza ("cheese", "pepperoni", "anchovies")
```

XCMDs, like any other command in hypercard, take their parameters on the same line as the command itself. XFCNs, like any other function in Hypercard, take their parameters in parentheses and return a value. This is the most important difference between XCMDs and XFCNs. You make the decision at link-time as to whether a code resource will be an XCMD or an XFCN. Please note, the code is *identical* for both XCMDs and XFCNs. Hypercard re-vectors ReturnValue for you depending on whether you invoked an XCMD or an XFCN.

The next field, *passFlag*, should be set to TRUE if you want Hypercard to pass the command on in the hierarchy after invoking the XCMD. Most of the time, you won't concern yourself with this field.

The next five fields in the parameter block are used for making *callbacks* to Hypercard [the subject of the next article - DK]. Callbacks allow you to invoke some hypercard commands from within an XCMD, quite a useful feature. An example of a callback is *GetFieldByNum* which will return a handle to the data in a field on the current card. This is analogous to "Get card field 1" in HyperTalk.

EntryPoint is set by Hypercard on entry to the XCMD and is used as a jumping off point for invoking callbacks. In effect, Hypercard will execute a jump instruction to EntryPoint for all callbacks. But before executing the jump instruction, Hypercard will place an integer value into the *Request* field. This integer tells Hypercard which callback to execute. The code at entryPoint will vector to the appropriate routine based on the value in Request.

Upon returning from a callback, Hypercard will have set the value of the *Result* field to some integer value to tell you how things went in the callback. A value of 0 indicates that no error occurred in the callback, 1 indicates that the callback failed for some reason or other and 2 indicates that the callback is not implemented.

The next to last field in the record, *inArgs*, is an array of up to eight input arguments to the callback. Although the arguments are passed as longInts (long in "C"), they may contain anything. Generally speaking, the Hypercard callback glue will handle type-casting for you.

Finally, callbacks can return up to 4 parameters in *OutArgs*. These parameters will be set by the callback and available to you from the calling XCMD. Callbacks are glued to the XCMD using standard Pascal interfaces so they're pretty easy to get along with.

Callbacks are a useful and powerful tool available to the XCMD programmer. Currently, Hypercard includes about thirty callbacks and I'll discuss each one in more detail next month.

The foregoing discussion implies that you'll mostly concern yourself with the first four fields in the record and let Hypercard manage the callback parameters. In practice, you'll mostly be concerned with *paramCount, params* and *returnValue.*

For the sake of our "C" readership, here's the definition of the parameter block as a "C" structure:

```
typedef struct XCmdBlock {
    short       paramCount;
    Handle      params[16];
    Handle      returnValue;
    Boolean     passFlag;

    char    *entryPoint;
    short       request;
    short       result;
    long        inArgs[8];
    long        outArgs[4];
  } XCmdBlock, *XCmdBlockPtr;
```

All Input and Output to an XCMD is passed through the parameter block. Armed with this information, here is a simple XCMD that does absolutely nothing (note: I use MPW Pascal and "C" in my examples).

```
{*****************************************}
{* File: SimpleXCMD.p                    *}
{*                                       *}
{* Shell for XCMDs and XFCNs in MPW Pascal  *}
{* ---------------------*}
{* In:   paramPtr = pointer to the XCMD    *}
{*       Parameter Block                 *}
{*                                       *}
{* ---------------------*}
{* © 1988, Donald Koscheka               *}
{* ---------------------*}
{*****************************************}

{*****************************
        BUILD SEQUENCE
pascal SimpleXCMD.p
link -m ENTRYPOINT -rt XCMD=6555 ∂
    -sn Main=SimpleXCMD ∂
    SimpleXCMD.p.o ∂
    "{Libraries}"Interface.o ∂
    "{PLibraries}"Paslib.o ∂
    -o "{xcmds}"your_stack_here
```

```
*******************************)

{$S SimpleXCMD }

UNIT Donald_Koscheka;

INTERFACE

USES
    MemTypes, QuickDraw, OSIntf, ToolIntf,
     PackIntf, HyperXCmd;


PROCEDURE EntryPoint( paramPtr: XCmdPtr);

IMPLEMENTATION

{$R-}
TYPE
    Str31 = String[31];

PROCEDURE SimpleXCMD(paramPtr: XCmdPtr); FORWARD;

{————EntryPoint————}
PROCEDURE EntryPoint(paramPtr: XCmdPtr);
    BEGIN
        SimpleXCMD(paramPtr);
    END;

{——SimpleXCMD————}
PROCEDURE SimpleXCMD(paramPtr: XCmdPtr);
VAR
    i : INTEGER;

{$I XCmdGlue.inc }

    BEGIN
        WITH paramPtr^ DO
            BEGIN
                returnValue := NIL;
            END;
    END;

END.
```

The Build sequence for this file is included in the header for the sake of convenience. After the compilation, we need to link the code into a resource and add it to some stack. First, let's take a closer look at the link options. The "-m ENTRYPOINT" option tells the linker that the first line of executable code in the resource is at the label ENTRYPOINT. Next, the "-rt XCMD=6555" option tells the linker that this file whould be written as a resource of type "XCMD" and should be assigned a resource ID of 6555. Because we are writing the resource directly out to the stack named "your_stack_here", any XCMD with ID 6555 will be overwritten by this link. To link the code as an XFCN, use "-rt XFCN=6555". That's about the only difference between XCMDs and XFCNs!

You must remember to note that resource types are case sensitive. Telling the linker to set the resource type to "xcmd" will work fine, only Hypercard won't recognize the resource as an XCMD. As far as numbering XCMDs goes, I don't know of any rational system that's been implemented yet. Follow the number guidelines provided in Inside Macintosh, Chapter five.

The option "-sn Main=SimpleXCMD" tells the linker to change the segment name of the main segment from "Main" to "SimpleXCMD". "SimpleXCMD.p.o" is the name of the input file, the library includes follow. The last line "-o your_stack_here" instructs the linker to add this XCMD to the resource fork of the stack whose name is "your_stack_here". Remember to include pathnames in your build.

The build sequence for XCMDs is pretty much a boiler-plate. You'll change type from XCMD to XFCN, the ID to whatever you want, and the input and output lines. About the most important addition you might make to the build is to add libraries as needed.

The XCMD itself follows the standard layout for a Pascal Unit. The unit name is inconsequential to us. A lot of programmers use "UNIT dummyUnit" to remind themselves of that . I use my name instead.

The interface portion is straightforward. Tell the compiler what files you want to include and declare the interface to your routine. Note that we use "EntryPoint" and not "SimpleXCMD" as the main entrypoint for this routine. I'll leave it as an exercise to the student to figure what that's all about.

Note the use of the {$R-} directive in the implementation section. This turns off range checking and results in more efficient code. Although we don't use the Str31 type declared in the TYPE statement, it is needed by the callbacks so we MUST include it here.

Procedure SimpleXCMD contains the actual code for the XCMD. Note that it takes exactly one parameter, a pointer to an XCmdBlock. Although the VAR statement is not used in the body of the code, I included it here so that those of you that are less fluid in Pascal can clearly see that the {$I XCmdGlue.inc} directive follows the CONST, TYPE and VAR declarations within SimpleXCMD. XCMDGlue contains the glue routines for the callbacks. They are simple compiled with the rest of the code. Because of the scoping of procedures feature of Pascal, the glue routines will be able to access paramPtr directly, you won;t need to pass it to each routine in turn.

Finally, the body of the program. Here we simply set the returnValue to NIL and exit. If you already have code that you want to implement in an XCMD or XFCN, replace the body of SimpleXCMD with the body of your routine and off you go!

```
PROCEDURE SimpleXCMD(paramPtr: XCmdPtr);
VAR
    i : INTEGER;

{$I XCmdGlue.inc }
BEGIN
    WITH paramPtr^ DO
        BEGIN
            returnValue := NIL;
        END;
END;
```

XCMDs in "C" are different enough in implementation to bear some discussion. Here is SimpleXCMD in "C":

```
/*****************************************\
*   file:    SimpleXCMD.c                 *
*                                         *
*  XCMD shell in MPW "C"                  *
```

```
*                                                *
*   C -q2 SimpleXCMD.c                           *
*   link -sn Main=SimpleXCMD ∂                   *
*   -sn STDIO=SimpleXCMD ∂                       *
*   -sn INTENV=SimpleXCMD -rt XCMD=3Ø1∂          *
*    SimpleXCMD.c.o -o your_stack_here           *
*                                                *
*   If you use parts of the "C"     *
*   Library, use this               *
*   link instead:                                *
*                                                *
*   link -sn Main=SimpleXCMD -sn ∂               *
*   STDIO=SimpleXCMD ∂                           *
* -sn INTENV=SimpleXCMD -rt XCMD=3Ø1∂            *
*    -m SIMPLEXCMD SimpleXCMD.c.o ∂              *
*   "(CLibraries)"CRuntime.o ∂                   *
*   "(CLibraries)"CInterface.o ∂                 *
*   -o your_stack_here                           *
*                                                *
* ____________________ *
* By:    Donald Koscheka                         *
* Date:  21-Sept-87                              *
* © Copyright 1987, Donald Koscheka              *
*                                                *
* ____________________ *
\********************************/

#include <Types.h>
#include <OSUtils.h>
#include <Memory.h>
#include <Files.h>
#include <Resources.h>
#include "HyperXCmd.h"


pascal void SimpleXCMD( paramPtr )
    XCmdBlockPtr paramPtr;
/*******************************
* SimpleXCMD()
*******************************/
{
    paramPtr->returnValue = nil;
}

#include <XCmdGlue.inc.c>
```

SimpleXCMD() is defined as a Pascal Void to tell the "C" compiler to push the parameters "Pascal Style". This means that parameters are pushed from left to right rather than from right to left. Also, we need to inform "C" that this function does not return a value, so we qualify the type with "void". Otherwise, "C" will leave room on the stack for an integer-wide return value.

An important difference between the two languages is that "C" does not allow scoping of procedures. The callback glue routines are not local to SimpleXCMD as is the case in Pascal. For this reason, when calling glue routines from "C", the first parameter passed must be a pointer to the XCmdBlock.

When you start programming XCMDs, you may encounter a seemingly nebulous link error, "No Data Initialization". The linker is telling you that there is no global memory to initialize for the XCMD. XCMDs are code resources, they are designed to be called "subroutine" fashion from Hypercard and as such do not have access to their own globals. Put another way, Hypercard owns the global pool from which XCMDs may draw. This means that the only kind of data that can be declared in an XCMD is local data, also known as automatics. Automatics get created on the stack and exist only for the life of the XCMD. When the XCMD returns to Hypercard, the local memory pool goes away.

You are most likely to encounter a problem with this when using strings in "C". Consider the following code fragment:

```
char *myStr;

myStr = "Colleen";
```

In this code, I declare a string pointer called *myStr*. I then point this string pointer at the properly formed string 'Colleen'. This code will compile correctly but the linker will not be able to work with it. When "C" compiles strings in-line, it actually puts the string into the global pool and points myStr at the string in global memory. Since the XCMD is assembled without global memory, the linker won't know what to do with this code. Pascal does not suffer this fate because it does not put the string into global memory. The following code will compile and link just fine in Pascal:

```
myStr    : StrPtr;

myStr  := 'Margaret';
```

The reason this works in Pascal and not in "C" is that Pascal tacks the string "Margaret" onto the end of the code resource, rather than put the string into global memory.

The upshot of this diatribe is don't declare global variables from XCMDs. If you need to use strings in "C", you'll need to hard-code the assignments:

```
myStr[0] = 'C';
myStr[1] = 'o';
myStr[2] = 'l';
myStr[3] = 'l';
myStr[4] = 'e';
myStr[5] = 'e';
myStr[6] = 'n';
myStr[7]= '\0'
```

If your going to be using a lot of strings, I would suggest that you either put them in resources (yuck!) or use Pascal instead.

This article presented the basics of XCMD programming and describes how to interface your code to Hypercard. Next month I'll introduce the callbacks and give some examples how they can make XCMD programming easier and more fun. If you're already an experienced Macintosh programmer, the above is information enough to get you started on XCMDs. If you're just getting started, this article should be just enough to help you get started, without being too much, to get you lost.

HyperChat™

HyperTools For HyperProgrammers

Fred Stauder
HyperChat Editor
Zurich, Switzerland
MacTutor Vol. 4 No. 7

on HyperChat

— HyperEditorial

This month we have a section on new tools that will aid development of your Hypercard Stacks. Also Don Koscheka will bring you another XCMD Corner where he shows how to do callbacks.

—ScriptTips

Last month I brought you AutoscriptEdit-II which allowed you to lock and unlock fields, to put up the tool windows, and to edit the script of an unlocked field. Some of you may have come across a bug which was a bit of a problem. When you "Tabbed" from field to field if the mouse was higher than the field the edit script would be invoked. What I have done in version III is to enable the edit script only with the option key. It seems that it is a current limitation of Hypertalk that I can't get around it in a neater way. I will be trying for a fix in Hypertalk to correct this. Another neat thing that I have added to version III is a feature that will allow you to "hide" Hypercard when you are in MultiFinder. First click in the top left corner of the card, this moves the card window up and left so that only a small section is still showing.

To return the window to the original position click in the small section of hypercard remaining. This should work in most stacks if you have AutoscriptEdit-III in your home stack.

For the occasions where it does not work, if there is a button over it etc., simply go home and then click in the corner. Next month I will bring you a really exciting extension to AutoScriptEdit and possibly in a self-installing button which will give you a toggle in your home stack such as "Power Keys".

```
——— AutoScriptEdit-III by Fred Stauder  June 1988 ———
- Put this Script into your Home Stack
- To use keep mouse btn down over Button or Field and move mouse
up
- To edit Bkgnd Script keep mouse btn down move mouse to -left of
card
- Card Script - move right
- Stack Script - move up
- Now works with unlocked fields
- To lock and unlock fields option click in them
- To show/hide tools/patterns at cursor
- hold option click on non-field areas
- bug fixed tab fields, needs option-move up on unlocked fields
- click in top left corner to hide card window, botom right to show

on mouseStillDown
  global CWLoc
  if the optionkey is down
  then
    if the target contains "field" then set locktext of target to
false
    else
      if the optionkey is down then
        set the visible of pattern window to not the visible of
pattern window
        set the loc of pattern window to the mouseloc
        set the visible of tool window to not the visible of tool
window
        set the loc of tool window to the mouseloc
        exit mouseStillDown
      end if
    end if
  end if
  if the mouseloc is within "0,0,10,10"
  then
    put the loc of card window into CWLoc
    set loc of card window to -504,-315
    exit mousestilldown
  end if
  if the mouseloc is within "500,333,512,346"
  then
    set loc of card window to CWLoc
    exit mouseStillDown
  end if
  put char 6 of the target into C6
  if C6 is quote then AutoScriptEdit
  if C6 is "i" then AutoScriptEdit
  if C6 is quote then exit mouseStillDown
  if C6 is "i" then exit mouseStillDown
  if the mouseV <item 2 of rect of target then edit script of target
  pass mouseStillDown
end mouseStillDown

on AutoScriptEdit
  if the mouseH < 2   then edit script of this bkgnd
  if the mouseH > 510 then edit script of this card
  if the mouseV < 2   then edit script of this stack
end AutoScriptEdit

on openfield
  if the optionkey is down
  then
    if the mouseV <item 2 of rect of target
    then
      edit script of target
      set locktext of target to false
      exit openfield
    end if
    set locktext of target to true
  end if
  pass openfield
end openfield

Non-Comercial use only - All Rights Reserved
Fred Stauder, Hypercard Editor, (HyperChat) MacTutor
```

-HyperTest

HyperTools for the HyperHacker

Softworks Inc. has released two tools to help develop stacks. Hypertools #1 is mainly aimed at stack developers and Hypertools #2 helps with formatting and customizing stacks.

Each stack contains 16 tools, four tools are common to both; scan tools, font tools, stack stats, and free space.

The presentation of Hypertools is very good and shows how a product can be made from individual smaller subunits.

© The Definitive MacTutor, Vol. 4

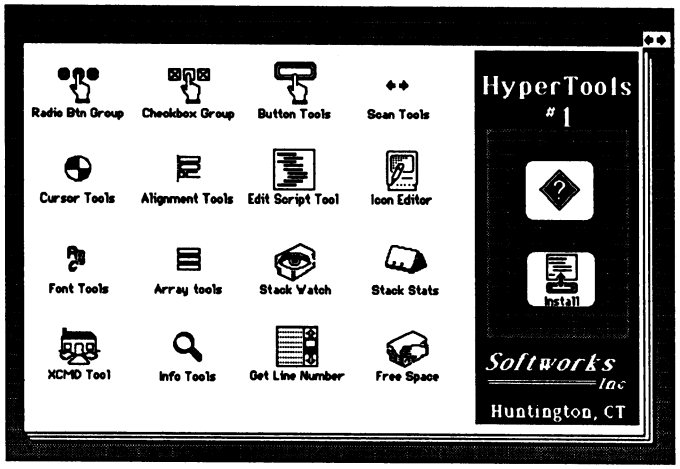


figure 1. HyperTools#1

The interface is very good with a consistent style throughout.

HyperTools incorporates a feature they called "scanning". When you click on a scan icon you can move between various cards in your stack by simply moving the mouse left to scan through previous cards; or to the right to scan through following cards. To stop at a card you click the mouse button.

You can perform scanning at two speeds: Normal Speed, when no keys on the keyboard are depressed; or Faster when the Shift Key on the keyboard is depressed. You can also automatically sequence through all of the cards by holding down the Option Key, the Scan tool will then continue sequencing through each card (left or right) until you release the option key.

Whenever you are scanning backwards (moving the mouse left), the cursor will change to an outlined arrow pointing left, a right arrow when going forward (moving the mouse right) and an arrow going in both directions when the mouse is not moved.

A well set out tool is the neat icon editor that integrates the best parts of Hypertalk and XCMD's.

The Icon Editor presents a list of all of the accessible icons in the current stack, Hypercard and the System file. By clicking on an entry in the field list, you can see the actual icon. By selecting the Edit Button, you can modify an existing icon. You can also Clone (make a new copy of) an existing icon, or create a brand new icon. When editing an icon, all of Hypercard's drawing features are available. You can even paste in a picture from Hypercard or the scrapbook to create the icon. This will save time going to ResEdit and back.

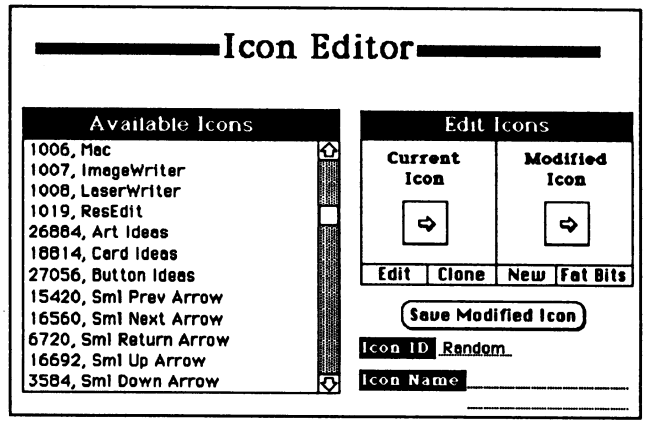


figure 2. Icon Editor Tool

Radio Button and Checkbox Group tools:

The Radio Button Group Tool lets you create the scripts for radio buttons and checkboxes on a card or background. The scripts for the radio buttons allow you to click on one button of the group and turn off all other buttons of the group automatically. The checkbox tool is similar.

The Button Tool gives you a pallet of common Buttons with their scripts. Not very exciting but still useful. What would be good is if you could add your own buttons to this tool.

The Cursor Tool shows all the available cursors, a cursor editor would be interesting.

The Alignment Tool allows you to select multiple cards or background fields and buttons and perform various repositioning actions on them as a group. You can also align and move elements to a grid which you can define.

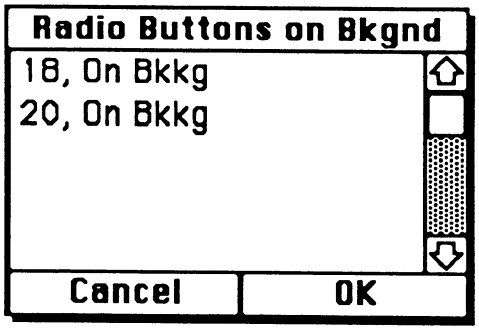


figure 3. Radio Button Tool

This is a well thought out tool and is very useful.

The Edit Script Tool is a button that you push before you click on the button to edit the script.

The Font Tool allows you to select multiple fields and/or Buttons on a card or background and change the font, the size, style, and the text alignment of selected items. You can quickly change the characteristics of all selected items.

The Array Tool allows you to add automatically rows and columns of fields or buttons at one time. This is a great little tool that saves time when you have to do repetitive creations.

The Stack Watch Tool adds commands to your stack script. This code is executed each time you run the stack. It will notify you whenever the free size of the stack exceeds 100K. Then it asks you if you want to compact.

The Stack Stats Tool quickly summarizes information about the current stack including the number of backgrounds, cards, and buttons and fields on cards and backgrounds. An example is shown on the next page

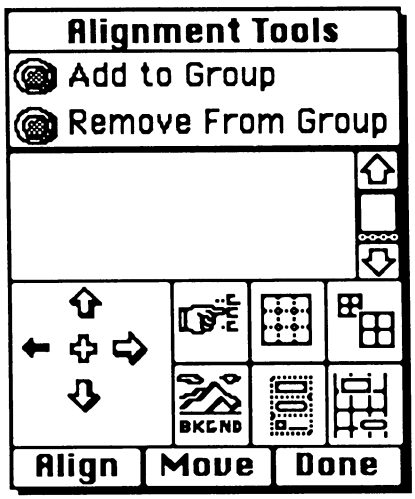


figure 4. Alignment Tool

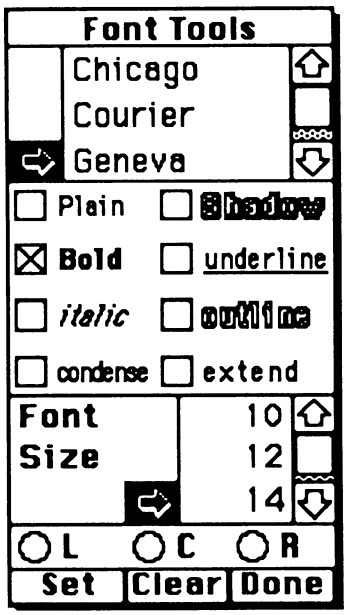


figure 5. Font Tool

The XCMD Tool quickly summarizes all of the accessible XCMD and XFNC that are installed in the current stack and Hypercard.

The Info Tools allows you to create various cross reference listings of the current stack. The listings created by these tools are displayed on the screen. You can also choose to print them or export them to a file on disk for use with a word processing program. You can create a list of all message handlers, references to a global variable or search for any string. For these functions, you can choose to search all scripts or choose from any combination of: the stack script, background scripts, background button and background field scripts, card scripts, card button or card field scripts.

If you search scripts for a string, a list detailing the number

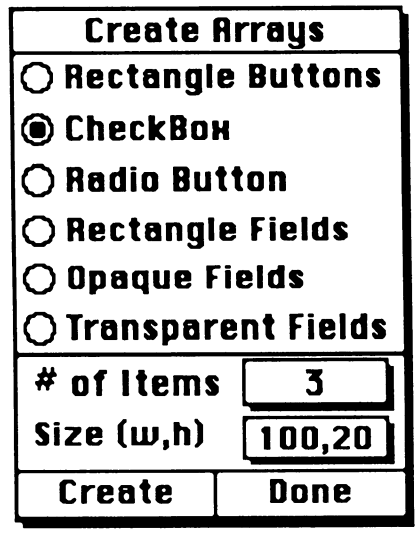


figure 6. Array Tool

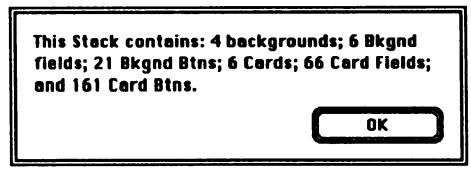


figure 7. Stack Stats Tool

of occurrences will be displayed on the screen.

You can also create a list of all card buttons, background buttons, card fields or background fields. For these functions you can choose to do the entire stack or a specific card.

The listings created by these tools are displayed on the screen. You can also choose to print them by selecting the Print button on the screen. You can also export a listing you created to a file on disk for use with a word processing program.

The Get Line Number Tool, lets you automatically create code,for a scrolling field on a card or background which returns the line number when the user clicks in it.

The Free Space Tool tells you how many bytes can be compressed from the current stack and allows you to compact the stack automatically.

Tools in HyperTools #2

The Choice List tool allows you to assign a list of selectable entry choices to a card or background field. Each stack may have up to 42 different choice categories. Clicking on the "Link a Choice List" icon will display the current list of all available categories. You can modify, delete or create categories from the window which is displayed. To add a choice list to a card or background field, click on a category from the window.

The Field Validation Tool lets you create a script for a card or background field which verifies that the information entered is a valid Hypercard Number or Date. You can also define whether any field on the card or background is required during entry.

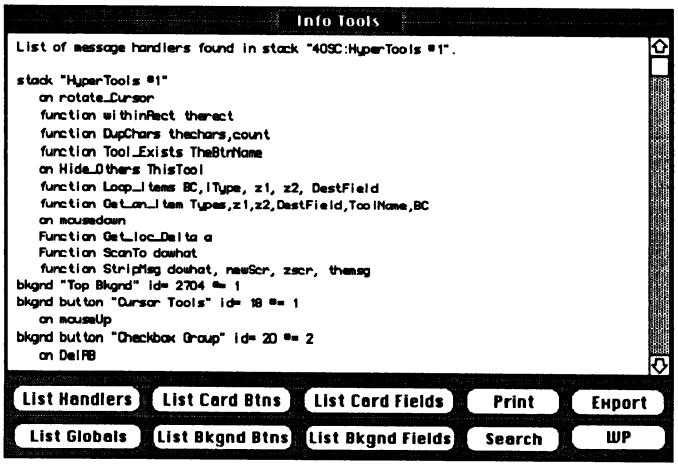


figure 8. Info Tool

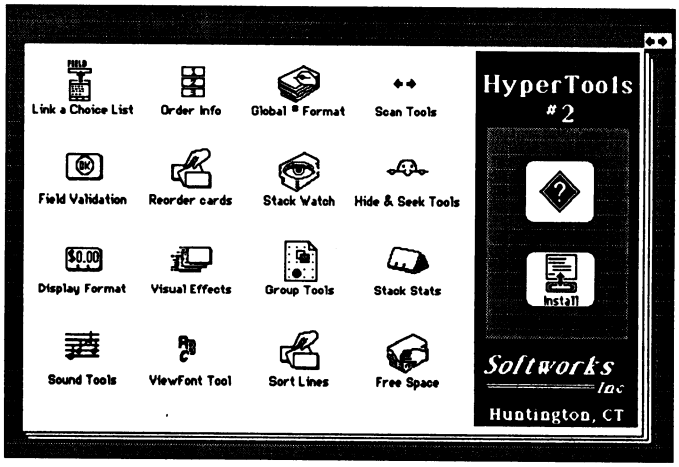


figure 9. HyperTools #2

The Display Format Tool allows you to add a script to a card or background field which will reformat the content of a card field whenever it is modified. You can select from the following display formats: number, date or text. This tool first allows you to scan to the card where the tool is to be installed.

The Visual Effects Tool displays a screen with buttons representing the various special effects that can be added to cards of your stack.

The Group Tools allow you to select multiple cards or background elements and perform various repositioning, copying and deletion activities on them as a group. This is very similar to the array tool in HyperTools 1.

The Global # Format Tool allows you to specify Hypercard's default precision for numbers and mathematical activities. When you click on this icon, a dialog showing the default numeric precision value is displayed.

The Sound Tools allow you to create and modify musical tunes. Sounds available in the current Stack and Hypercard are displayed in a scrollable list. Clicking on an entry from the list will change the current tune settings to use the selected Sound. You can create tunes using the piano keyboard which is displayed

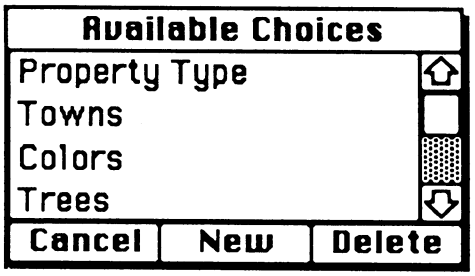


figure 10. Choice List Tool

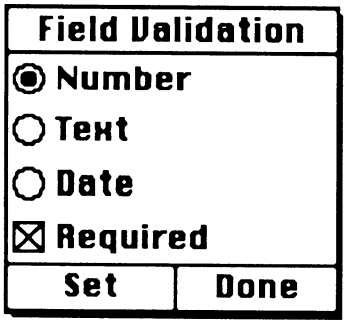


figure 11. Field Validation Tool

on the top of the screen and play them with any Sound and at any tempo you specify. You can also directly modify a tune using the editable field provided.

The ViewFont Tool allows you to view the various Fonts in

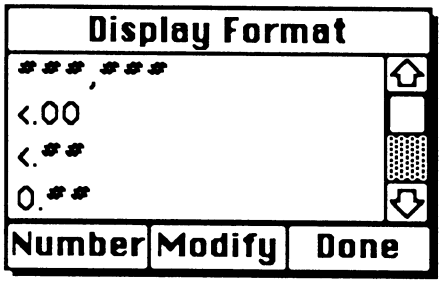


figure 12. Display Format Tool

the current Stack, Hypercard or your System File. You can sequence through all of the fonts one at a time or all at the same time using the Prev Font and Next Font buttons. You can change or modify the text in the scrollable areas to see how it looks in various fonts. To restore the original text just click Restore Text. You can change the font size with the Change Font Size button.

The Hide & Seek Tools allow you to see fields and buttons on cards and backgrounds even if they are hidden. This tool first allows you to scan to the card where the tool is to be installed.

The Reorder Cards Tool allows you to reorder all or a portion of the cards in the current stack. This tool allows you to scan through various cards and select the order of each card by clicking on the card.

The Order Info Tool allows you to see a list of all of the card field or card buttons. By selecting the appropriate buttons you can see an ordered list of all of the buttons or fields on the card or

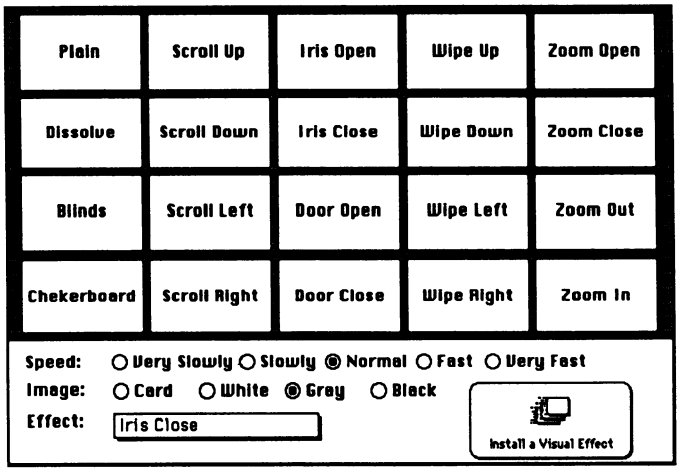


figure 13. Visual Effects Tool

background.

The Sort Lines Tool allows you to sort the contents of a scrollable field (or other multi-line field).

HyperTools includes a built-in help function that lets you quickly get information on the different tools. To get help on any

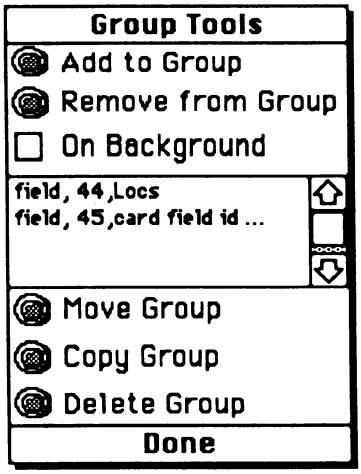


figure 14. Group Tool

tool on this screen, click on the help icon. Then click on any of the icons for the tools shown.

In Summary the HyperTools package is a useful development package for the novice and a time saving device for the more experienced HyperProgrammer.

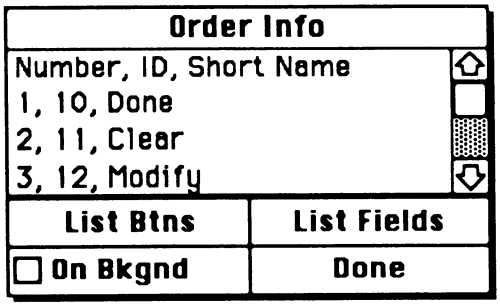


figure 18. Order Info Tool

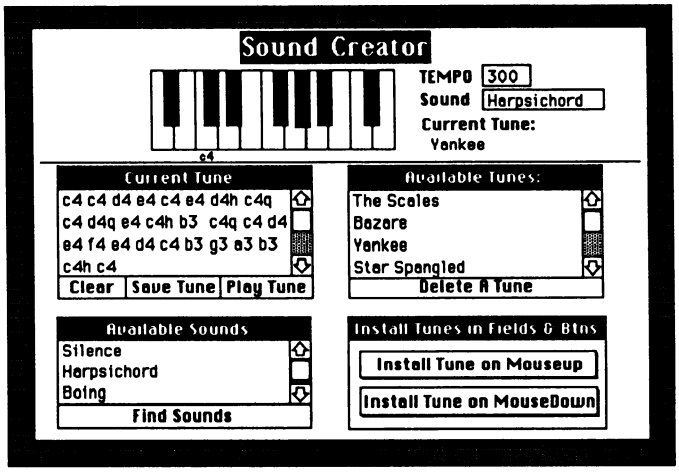


figure 15. Sound Tool

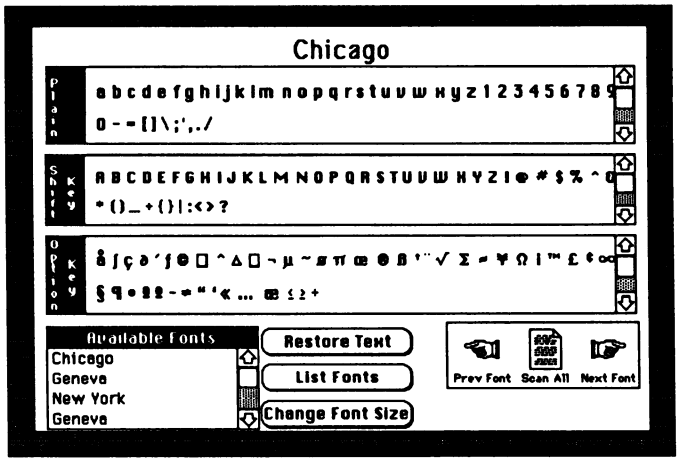


figure 16. View Font Tool

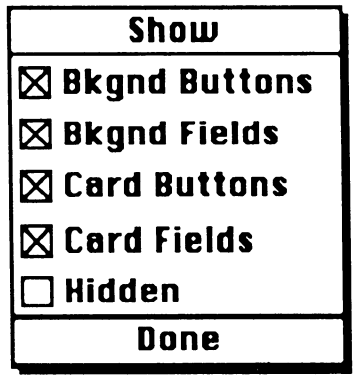


figure 17. Hide & Seek Tool

© The Definitive MacTutor, Vol. 4

HyperChat™

XCMD Corner

Donald Koscheka
Apple Computers, Inc.
MacTutor Vol. 4 No. 7

—Don Koscheka's XCMD Corner

Last month I introduced XCMD programming. I explained the parameter block and discussed the interface between HyperTalk and the code that you write in either Pascal, "C" or the language of your choice. If you're an experienced Macintosh programmer, that's enough information to get started. The designers of HyperCard didn't stop with just defining the interface for you. They went beyond what might be reasonably expected and provided some HyperTalk programming capabilities to the XCMD programmer. The XCMD programmer gains access to these capabilities through *callbacks*. Callbacks are procedures and functions that you call from your XCMD. Callbacks literally jump into Hypercard to perform some function.

Once you get the hang of XCMD programming, you'll come to rely on some of the callbacks quite frequently. For example, Pascal programmers are accustomed to strings that are preceded by a length byte and that have a maximum of 255 characters (the Str255 type in Pascal) while HyperCard uses strings that have no length byte and are terminated with zero (referred to as Zero-strings, zero-terminated strings or "C" strings). HyperTalk provides callbacks to convert from zero-terminated strings to Pascal and back again. The XCMD programmer can also use callbacks to retrieve and set the contents of fields and global containers as well as to send messages back to Hypercard.

One of the reasons that this access to HyperCard containers is so important is that XCMDs do not have access to the application globals. When Hypercard starts up, it takes control of its application globals and heap just as any application would. With Hypercard in control of the heap, your XCMD becomes a "guest" of Hypercard. You don't have access to the globals from your XCMD so you'll need a safe place to store information that you want to keep around.

A good candidate is a global container. Although Hypercard itself prefers to see text strings in containers, it's not particular about what you put into a container; you can use the SetGlobal callback to put data into a global container, and GetGlobal to retrieve that data. Make sure that you declare any global containers in Hypercard before accessing them in an XCMD. Your XCMD may need to be backward compatible earlier versions of Hypercard that bombed if you called SetGlobal with an undeclared container name.

An interesting example of the use of callbacks is to send a card message back to HyperCard from a callback. The XCMD in listing 1, "SendMeAMessage", takes one parameter which is the message to send back to HyperCard. Since messages are sent back as Pascal-format strings, we must convert the input string into a pascal string and then call SendCardMessage to send the message. As a matter of form, we set the return value to NIL indicating that this XCMD doesn't have a result code (Hypercard will interpret NIL as *empty*).

The first callback that SendMeAMessage invokes, *ZeroToPas*, converts input parameter 1 from a zero-terminated string to a pascal-string. Input parameters are passed as handles so the parameter needs to be dereferenced one-time to convert the value to a pointer. ZeroToPas also expects you to pass a reference to the pascal-string into which you'll store the converted Pascal-string. The second callback, *SendCardMessage*, sends the message back to Hypercard. To invoke this XCMD from a script use the form:

```
SendMeAMessage "go next card"
```

Because Hypercard already handles message passing, this XCMD may not seem terribly useful. Nonetheless, it is a good illustration of callbacks and the technique is useful if you need to alert the user that some asynchronous event has completed.

```
{*******************************}
{* File: SendMeAMessage.p
    *}
{*******************************}
{*******************************
    BUILD SEQUENCE
       (IGNORE LINK WARNINGS)
pascal SendMeAMessage.p
link -m ENTRYPOINT -rt XCMD=65534 ∂
  -sn Main=SendMeAMessage ∂
  SendMeAMessage.p.o ∂
  "(Libraries)"Interface.o ∂
  "(PLibraries)"Paslib.o ∂
  -o "(xcmds)"testxcmds
*******************************}

{$S SendMeAMessage }

UNIT Donald_Koscheka;

{————INTERFACE————}
INTERFACE

USES MemTypes, QuickDraw, OSIntf, ToolIntf,
  PackIntf, HyperXCmd;

PROCEDURE EntryPoint( paramPtr: XCmdPtr);

{——IMPLEMENTATION——}
IMPLEMENTATION
{$R-}
```

© The Definitive MacTutor, Vol. 4

```
TYPE
  Str31    = String[31];

PROCEDURE SendMeAMessage(paramPtr:XCmdPtr);
          FORWARD;

{------- EntryPoint -------}
PROCEDURE EntryPoint(paramPtr: XCmdPtr);
  BEGIN
    SendMeAMessage(paramPtr);
  END;

{------- SendMeAMessage -------}
PROCEDURE SendMeAMessage(paramPtr: XCmdPtr);
VAR
  theMessage : Str255;

{$I XCmdGlue.inc }

BEGIN {*** Body of XCMD ***}
  ZeroToPas( paramPtr^.params[1]^, theMessage );
  SendCardMessage( theMessage );
  paramPtr^.returnValue := NIL;
END; {*** Body of XCMD ***}
END.
```

Listing 1. SendACardMessage

XCMDs can also use callbacks to get the contents of a field or a global container. Listing 2, GetHomeInfo, uses two new callbacks (1) *GetFieldByNum*, to get the contents of background field 1 on the home card and (2) *SetGlobal* to set the contents of some global container.

GetHomeInfo Takes one parameter, the name of the global container to set. First, convert the parameter to a pascal-string. Next, use a series of callbacks to push the current card and go to the home stack. Once we get to the home stack, we call GetFieldByNum to get field data. The first parameter that GetFieldByNum takes is set to TRUE if you want to retrieve the contents of a card field and set to FALSE to retrieve the contents of a background field. The second parameter is the number of the field to retrieve. Alternatively, you could use *GetFieldByID* if you knew the id of the field or *GetFieldByName* if you knew the name.

GetFieldByNum returns a handle to the zero-terminated data that was stored in background field 1 of the home card. We then invoke SetGlobal to set the contents of the global container to whatever is stored in fieldData. Finally, we pop the current card to get back to where we started. A typical invocation of this XCMD is:

```
global myData
GetHomeInfo "myData"
```

```
PROCEDURE GetHomeInfo(paramPtr: XCmdPtr);
VAR
  globalName : Str255;
  fieldData  : Handle;

{$I XCmdGlue.inc }

BEGIN
  WITH paramPtr^ DO
  BEGIN
    ZeroToPas( params[1]^, globalName );
    SendCardMessage( 'Push Card' );
    SendCardMessage( 'Go Home');

    fieldData := GetFieldByNum( FALSE, 1 );
    SetGlobal( globalName, fieldData );

    SendCardMessage( 'Pop Card');
    returnValue := NIL;
  END;
END;
```

Listing 2. Get Home Info.

So far I've showed you XCMDs that don't do anything that you can't already do in HyperTalk. A key feature of XCMD programming is that you can write your own commands to augment or add capabilities to Hypercard. Listing 3, "FCreateXFCN", lets you create a file of any type from HyperCard. This XFCN demonstrates how XCMDs provide greater access to the toolbox than is available to the HyperTalk script writer.

FCreateXFCN introduces two new callbacks. *NumToStr* returns a result code to the script. NumToStr converts a signed long integer to a pascal-format string. If you don't want a signed entity, use *LongToStr* instead which will convert the long integer without regard to sign.

The second callback introduced in this XFCN is *PasToZero* which takes a pascal format string and returns handle to a zero-terminated string. PasToZero is a handy way of copying from a pascal-string back to a zero-terminated string to return text to Hypercard.

I wrote the XFCN in "C" to show the difference in formats between Pascal and "C" XCMDS and to provide a template for "C" programmers. An important difference is that the parameter list, params, starts at index 0 for "C" and index 1 for Pascal.

FCreateXFCN first converts parameter 1 (params[0]) into a pascal-string and then moves the first four bytes of parameters 2 and 3 into the variables creator and type respectively. These two variables are of type OSType which is a special Macintosh type containing four consecutive ASCII characters. All four characters in this type are significant.

The result code will be empty if no error occurred and the file was created properly, otherwise it will return the OSErr number that occurred. First, convert the result code back to a pascal-string and then call PasToZero to convert that string into a handle to a zero-terminated string.

© The Definitive MacTutor, Vol. 4

The XFCN would be more useful if it returned a brief description of the error in English, but I think I'll leave that as an exercise for the student. Call FCreateXFCN with the following script:

```
Put FCreateXFCN( "New File Name", "WILD", "STAK" )
```

The first parameter is the name of the file you wish to create, the second parameter is the creator and the last parameter is the file type. The foregoing invocation will create an empty stack. What can you do with that?

```
/*****************************\
* file: FCreateXFCN.c
    *
\*****************************/

/*****************************
    BUILD SEQUENCE

C -q2 -g FCreateXFCN.c
link  -sn Main=FCreateXFCN ∂
  -sn STDIO=FCreateXFCN ∂
   -sn INTENV=FCreateXFCN ∂
   -rt XFCN=300 ∂
   -m FCREATEXFCN ∂
   FCreateXFCN.c.o ∂
   "{CLibraries}"CInterface.o ∂
   -o testXCMDs

*****************************/
#include <Types.h>
#include <OSUtils.h>
#include <Memory.h>
#include <Files.h>
#include <Resources.h>
#include "HyperXCmd.h"

pascal void FCreateXFCN( paramPtr )
  XCmdBlockPtr paramPtr;
/*****************************
* In:  Paramblock:
*    param[0] == filename
*    param[1] == TYPE
*    param[2] == CREATOR
* Out: result code in returnValue
*****************************/
{
char   *fname;
OSType type, creator;
Str31 str;
char   vName[33];
short    error, vRefnum;

/** coerce the volume reference
***
*** number from the system
**/
GetVol( vName, &vRefnum );

/** extract the filename from
***
*** the parameter list
  **/
fname  = *(paramPtr->params[0]);
BlockMove(*(paramPtr->params[1]),&creator,4 );
BlockMove(*(paramPtr->params[2]),&type,4 );

error = Create( fname, vRefnum,
    creator, type );

FlushVol( 0L, vRefnum );

if( !error )
  paramPtr->returnValue = 0L;
else{
  NumToStr( paramPtr, (long)error, &str );
  paramPtr->returnValue =
    PasToZero( paramPtr, &str );
  }
}

#include <XCmdGlue.inc.c>
```

Listing 3. FCreateXFCN

Although this article covered some of the more frequently used callbacks, my objective was to present you with the spirit of the callback mechanism. You should have no trouble using any of the callbacks that are currently defined in the HyperCard Developer's ToolKit (available from APDA). The most important lesson here is that before you go off and write an XCMD, check the callback list to see how you might incorporate them into your XCMDs. Try to get the callbacks to do as much work as possible for you so that you can concentrate on the code that you are trying to write.

Next Month: *SortList*, an XCMD that sorts a field by line.

end HyperChat

HyperChat™
Auto Hyper Edit Moves in Groups

Fred Stauder
HyperChat Editor
Zurich, Switzerland
MacTutor Vol. 4 No. 8

HyperCard

AutoHyperEdit

Over the last few months we have seen the evolution of Hypercard Stacks. We have also been evolving AutoScriptEdit by showing you how to increase the functionality while retaining simplicity. I have had a lot of feedback about ASE and most people prefer it to the inbuilt shortcuts in Hypercard 1.2.

So what was I to do for an encore, well I always wanted to group and move buttons etc. Souix Lacey did a great job with Groupies, and I wondered if I could make Hypercard act more in a Mac way. I decided to write AutoHyperEdit.

Well I started experimenting and I was thoroughly addicted to solving the problem. Even half way throug writing it I thought it would be impossible to get around some obstacles that would make it work. The script basically allows the user to shift and click on objects to group them. The problem was if you clicked on a button for example after my stack had done it's stuff it would then execute the button that was clicked on. So I couldn't use nice things like target. The way I got around that was to check what objects were under the mouse at the time of the shiftKey.

After I had the scripting finished I decided I would make a model Installer and Demo Stack. I will first take you through the Stack and then get into how and what was done.

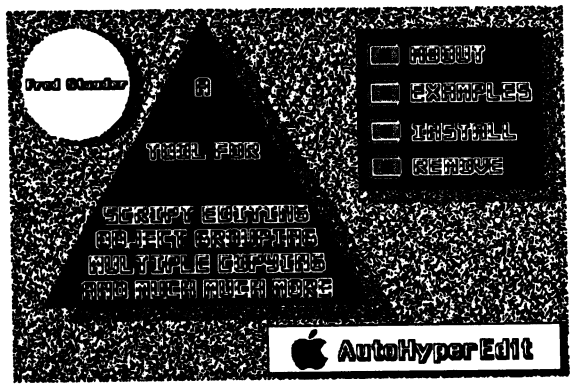


figure 1. AutoHyperEdit

The opening card shows what a little shading can do to add depth to your stack. The buttons I created to give as much feedback as possible. When you click on a button it appears to move in and the center black square turns to white. This is done by switching icons. Also to implement more feedback a click sound is heard.

I can't stress the importance of a good user interface particularly in Hypercard. At least half of my time in this project was spent on the interface. Apple is spending a lot of dollars on interface technology and have recognised the need for using it to aid in the smooth information flow between computer and machine. There is a group at Apple called the Human Interface Group, their sole job is to work on interface ideas and guidelines. If you have not read the Human Interface Guidelines do so immediately. It is a must. The guidlines are available through APDA and in some Bookshops and Computer Stores.

If you go to the examples card you will see three new buttons; "Script Tools", "Object Tools" and "Aligning Tools".

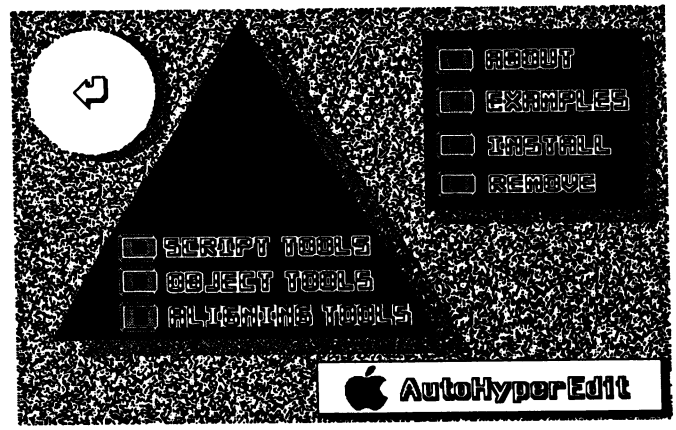


figure 2.

The Script Tools are the tools found in AutoScriptEdit-III. I use an interesting way to demo how the tools are to be used. There is part of a keyboard and part of a mouse pad and a mouse in the bottom left of the card, see figure 3.

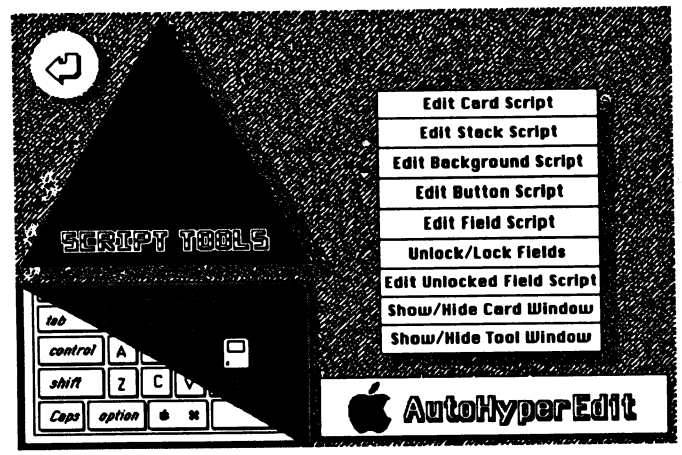


figure 3.

© The Definitive MacTutor, Vol. 4

When you click on a selection, you also get visual and auditory feedback, a field pops up along with two buttons; "Close" and "Show Me".

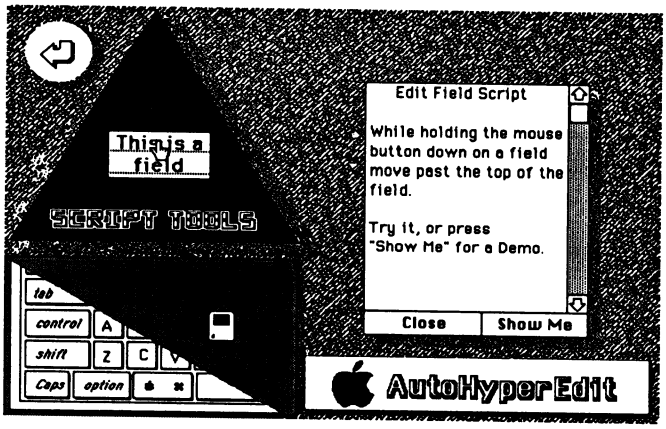


figure 4.

Clicking on "Show Me" Starts the demo. First the cursor dissapears and an icon that is identical to the cursor is shown at the point of interest. To attract the attention to the moving parts I "flash" (hilite/unhilite) both the cursor and the the mouse icon. now I play the tune:

play harpsichord tempo 1200 c5 c6 c7

This signifies the start of what you should do. I then play the word "mousedown" which has been digitized. The icon changes showing a mouse with the button down. I showed the original demo's to non Mac users and they could follow how to do it easily. My previous tests where I didn't have all the feedback the user was a bit lost and did not know where to focus his attention. The rest of the demo shows the "mouse" and the "cursor" moving up above a field. The script editor is then evoked.

The Object Tools shows buttons; "Grouping", "Moving", "Copying", and "Clearing" see figure 5.

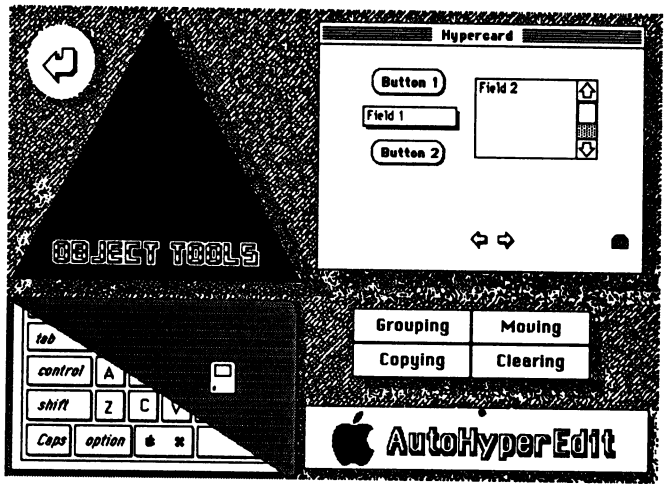


figure 5.

The action of the buttons remain consistent even though the location is slightly different. One of the things to watch out for in your stacks is to keep consistency and simplicity at the forefront of your mind when you are designing.

The demo of grouping shows the tool window first to show that you must be in the browse tool mode. You then click on an object with the shift key down. This will hilite a button.

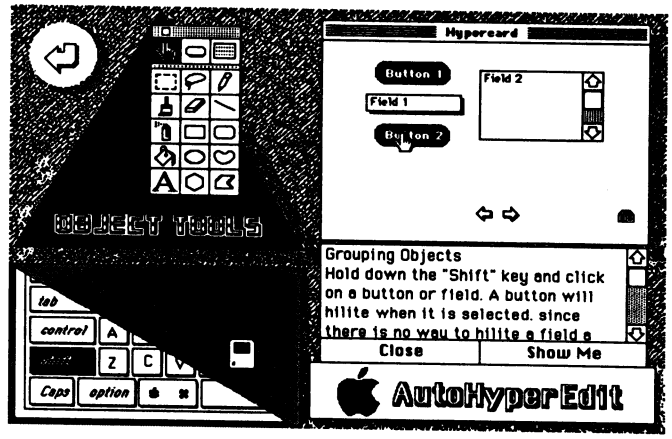


figure 6

There was a problem of not being able to hilite a field, so as well as a hilite I use a click tone. To select more objects just shift click again and to de-select you also shift click. Once you have finished selection you must double click not on an object. This will bring you to the button tool where you can choose a menu item such as "copy button". Or simply type command-C. I used a feature of Hypercard 1.2 to show the card picture. I have a menu as a card picture and I show it and drag a hilited transperent button to act as a regular menu, see figure 7. Now go to the card you want to paste to and do a command-V and the objects will appear.

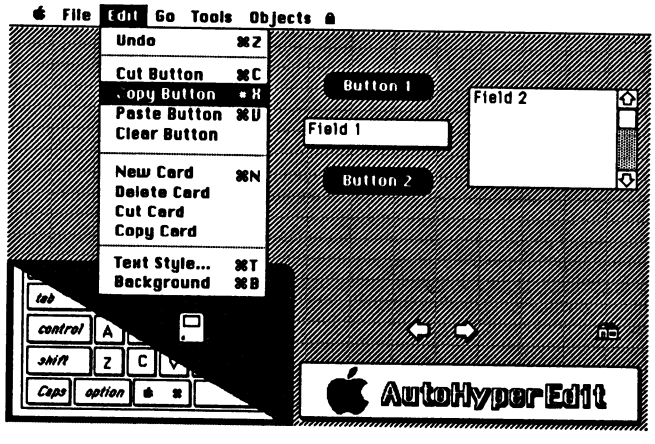


figure 7.

Clearing objects works in the same way, first selection then, double click, then menu command.

The next section is Aligning Tools. This allows you to select objects as before and then to hold down the mouse where you want the objects to be aligned, press the arrow key which corresponds to the alignment i.e. left arrow align left.

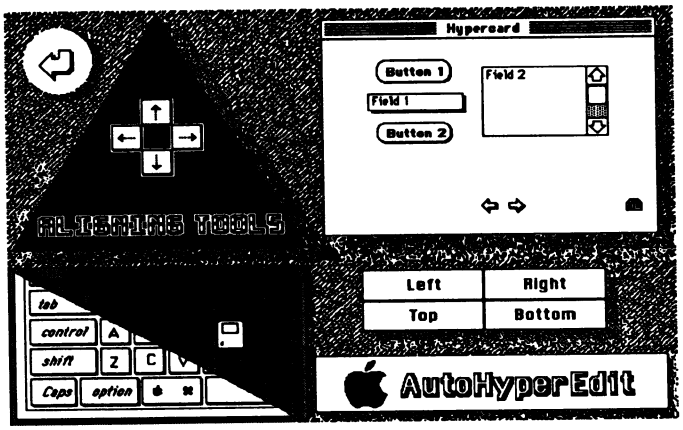


figure 8.

I also created an installer that puts the script into your home stack and adds a checkbox in the preferences card to allow you to turn AHE on and off. I do this in an interesting way. As well as the checkbox I add a hidden field to the card. I put the home stack into the field and search for specific comment strings that are on the same line as the handlers and I comment out the begining and end of the handlers.

```
on mouseUp
  if the hilite of me is false
  then
    put the script of this stack  into cd fld "TempAHE"
    lock screen
    show cd fld "TempAHE"
    repeat 8 times
      set cursor  to busy
      find "--**AutoHyperEdit**--"
      put the foundline into temp
      select temp
      put "--" before the selection
      put the selection into temp
    end repeat
    set script of this stack to cd fld "TempAHE"
    put empty into cd fld "TempAHE"
    hide cd fld "TempAHE"
    unlock screen
  else
    put the script of this stack  into cd fld "TempAHE"
    lock screen
    show cd fld "TempAHE"
    repeat 8 times
      set cursor  to busy
      find "--**AutoHyperEdit**--"
      put the foundline into temp
      select temp
      put the selectedtext into stext
      delete char 1 to 2 of stext
      put stext into the selection
      put stext into temp
    end repeat
    set script of this stack to cd fld "TempAHE"
    put empty into cd fld "TempAHE"
    hide cd fld "TempAHE"
    unlock screen
  end if
end mouseUp
```

The Remove card simply removes all traces of AHE.

Here is the script I will go through the important parts and tell you what they do.

```
-- AutoHyperEdit 1.0  by Fred Stauder  June 1988 --

- Put this Script into your Home Stack
- To use keep mouse btn down over Button or Field and move mouse
up
- To edit Bkgnd Script keep mouse btn down move mouse to -left
of card
- Card Script - move right
- Stack Script - move up
- Now works with unlocked fields
- To lock and unlock fields option click in them
- To show/hide tools/patterns at cursor
- hold option click on non-field areas
- click in top left to hide cd window botom right to show
-Group objects by shift clicking in browse tool
-Move objects by holding mouse down and draging
-Copy and Clear groups of objects after selecting by using
menus
- Align objects after selecting, hold mouse down at align loc
and press
- the arrow key i.e. align left use left arrow key
- Turn AHE on and Off using the  Checkbox installed in the Pref
Card of Home
```

The globals are CWSlist is where the selected objects id's are cwloc is the card window location to hide the card and selectcomp is a flag for when the selections are compleated.

Gobo resets the hilites and puts empty into CWSList. The first section is AutoScriptedit Stuff.

```
on mouseStillDown --**AutoHyperEdit**--
  Global cwslist,cwLoc,selectcomp
  if selectcomp is "true" then gobo

  put empty into CWSList

  if the optionkey is down
  then
   if the target contains "field" then set locktext of target
to false
    else
      if the optionkey is down then
       set the visible of pattern window to not the visible
of pattern window
        set the loc of pattern window to the mouseloc
        set the visible of tool window to not the visible of
tool window
        set the loc of tool window to the mouseloc
        exit mouseStillDown ——
      end if
    end if
  end if
  if the mouseloc is within "0,0,10,10"
  then
    put the loc of card window into CWLoc
    set loc of card window to -504,-315
    exit mousestilldown
  end if
  if the mouseloc is within "500,333,512,346"
  then
    set loc of card window to CWLoc
    exit mouseStillDown
  end if
```


© The Definitive MacTutor, Vol. 4

```
      put char 6 of the target into C6
      if C6 is quote then AutoScriptEdit
      if C6 is "i" then AutoScriptEdit
      if C6 is quote then exit mousestilldown
      if C6 is "i" then exit mousestilldown
     if the mouseV <item 2 of rect of target then edit script of
target
      else
        ————————CWS
        put empty into foundit
        put false into offdetected
        if cwslist is empty
        then
          if the shiftkey is down
          then
            repeat until offdetected is true
              if the shiftkey is down
              then
                shiftselect
```

Here we go to the routine that selects objects.

```
                put false into foundit
              else
                movegroup
                wait until the mouse is down
              repeat with y = the number of items in CWSlist down
to 1
                 if the mouseloc is within the rect of item y of
cwslist
                  then
                    put false into foundit
                    exit repeat
                  else
                    put true into foundit
                  end if
                end repeat
              end if
              if foundit is true
                then if the shiftkey is up then put true into
offdetected
            end repeat

            choose btn tool
            select item 1 of cwslist
            put "true" into selectcomp
          end if
        end if
      end if
      -pass mouseStillDown
    end mouseStillDown -**AutoHyperEdit**-


    on AutoScriptEdit
      if the mouseH < 2   then edit script of this bkgnd
      if the mouseH > 510 then edit script of this card
      if the mouseV < 2   then edit script of this stack
    end AutoScriptEdit

    on openfield -**AutoHyperEdit**-
      if the optionkey is down
      then
        if the mouseV <item 2 of rect of target
        then
          edit script of target
          set locktext of target to false
          exit openfield
        end if
        set locktext of target to true
      end if
      pass openfield
    end openfield -**AutoHyperEdit**-
```

This traps the menu selections so that copy buttons etc will work on groups of objects

```
    on domenu stuff -**AutoHyperEdit**-
      global WhatsMyCardName,CWSList,selectcomp
      if selectcomp is true
      then
        if  stuff is "clear button"
        then
          clrobj
          exit domenu
        else if stuff is "Copy Button" then
          put the long name of this card into WhatsMyCardName
          pass domenu
        else if stuff is "cut button" then put tip into top
        else if stuff is "paste button" then
          pasteobj
          exit domenu
        else if stuff is "Copy field"then
          put the name of this card into WhatsMyCardName
          pass domenu
        else if stuff is "paste field" then pasteobj
        else if stuff is "clear field" then clrobj
        pass domenu
        exit domenu
      end if
      pass domenu
    end domenu -**AutoHyperEdit**-
```

This is the Object clearing routine.

```
    on clrobj
      global CWSList
      lock screen
      repeat with p= the number of items in CWSlist down to 1
        set cursor to busy
        put item p of CWSlist into it
        select it
        type "x" with commandkey
      end repeat
      choose browse tool
      unlock screen
      put empty into Cwslist
    end clrobj
```

This is the object pasting routine

```
    on pasteobj
      global CWSList,WhatsMyCardName,selectcomp
      put false into selectcomp
      set lockscreen to true
      set lockmessages to true
      repeat with x= the number of items in CWSlist down to 1
        if second word of item x of CWSList ¬
        is "field" then choose field tool else choose btn tool
        set cursor to busy
        push card
        go to WhatsMyCardName
        put item x of CWSlist into it
       if second word of it is "btn" then set hilite of it to false
       if first word of it is "btn" then set hilite of it to false
        select it
        type "c" with commandkey
        pop card
        if first word of it is "Bkgnd" then type "b" with cmdkey
        type "v" with commandkey
       if first word of it is "Bkgnd" then type "b" with cmdkey
      end repeat
      choose browse tool
      set lockscreen to false
      set lockmessages to false
      put empty into CWSlist
      put empty into WhatsMyCardName
      put false into selectcomp
      choose browse tool
```

```
    end pasteobj

    on gobo
      global CWSList,turnoff,selectcomp
      repeat with x= the number of items in CWSlist down to 1
       if second word of item x of CWSList is "btn" then set the
hilite of item x of CWSlist to not the hilite of item x of CWSlist
       if first word of item x of CWSList is "btn" then set the
hilite of item x of CWSlist to not the hilite of item x of CWSlist
      end repeat
      put empty into CWSList
    end gobo
```

Here I had to test for different objects. I tried to put "if the mouseloc is within the rect of *a variable* then...." where the variable contained an objuct such as btn and another variable. This did not work it is a pity I could have cut the script down considerably. Because of this I have to handle the object types seperately.

```
    on shiftselect
      global CWSlist,w,x,y,z
      repeat until the shiftkey is up
        repeat while the mouse is down
          repeat with w = the number of cd btns down to 1
             if the mouseloc is within the rect of btn w then
HITCBtn
          end repeat
          repeat with x = the number of bkgnd btns down to 1
           if the mouseloc is within the rect of bkgnd btn x then
HITBBtn
          end repeat
          repeat with y = the number of cd flds down to 1
            if the mouseloc is within the rect of cd fld y then
HITCFld
          end repeat
          repeat with z = the number of bkgnd flds down to 1
           if the mouseloc is within the rect of bkgnd fld z then
HITBFld
          end repeat
        end repeat
      end repeat
    end shiftselect

    on HitBFld
      global CWSlist,z
      put false into foundit
      if the commandkey is down then set the visible of bkgnd fld
z to not the visible of bkgnd fld z
      if the number of items in CWSlist =0
      then
        put "Bkgnd field id " & the id of Bkgnd fld z & ","after
CWSList
        play harpsichord tempo 100 b#x7
        exit HitBFld
      end if
      repeat with p = the number of items in CWSlist  down to 1
        put item p of CWSList into it
        if it = "Bkgnd field id " & the id of Bkgnd fld z
        then
          delete item p of CWSList
          play harpsichord tempo 100 b#x7
          exit HitBFld
        end if
      end repeat
      put "Bkgnd field id " & the id of bkgnd fld z & ","after
CWSList
      play harpsichord tempo 100 b#x7
    end HitBFld

    on HitCFld
      global CWSlist,y
      put false into foundit
      if the commandkey is down then set the visible of cd fld y
to not the visible of cd fld y
      if the number of items in CWSlist =0
      then
       put "Card field id " & the id of cd fld y & ","after CWSList
        play harpsichord tempo 100 b#x7
        exit HitCFld
      end if
      repeat with p = the number of items in CWSlist  down to 1
        put item p of CWSList into it
        if it = "Card field id " & the id of cd fld y
        then
          delete item p of CWSList
          play harpsichord tempo 100 b#x7
          exit HitCFld
        end if
      end repeat
      put "Card field id " & the id of cd fld y & ","after CWSList
      play harpsichord tempo 100 b#x7
    end HitCFld

    on HitBBtn
      global CWSlist,x
      put false into foundit
      if the commandkey is down then set the visible of bkgnd btn
x to not the visible of bkgnd btn x
      set the hilite of bkgnd btn x to not the hilite of bkgnd btn
x
      if the number of items in CWSlist =0
      then
        put "bkgnd btn id " & the id of bkgnd btn x & ","after
CWSList
        play harpsichord tempo 100 b#x7
        exit hitBBtn
      end if
      repeat with p = the number of items in CWSlist  down to 1
        put item p of CWSList into it
        if it = "bkgnd btn id " & the id of bkgnd btn x
        then
          delete item p of CWSList
          play harpsichord tempo 100 b#x7
          exit hitBBtn
        end if
      end repeat
      put "bkgnd btn id " & the id of bkgnd btn x & ","after CWSList
      play harpsichord tempo 100 b#x7
    end HitBBtn

    on HitCBtn
      global CWSlist,w
      put false into foundit
      if the commandkey is down then set the visible of btn w to
not the visible of btn w
      set the hilite of btn w to not the hilite of btn w
      if the number of items in CWSlist =0
      then
        put "btn id " & the id of btn w & ","after CWSList
        play harpsichord tempo 100 b#x7
        exit hitCBtn
      end if
      repeat with p = the number of items in CWSlist  down to 1
        put item p of CWSList into it
        if it = "btn id " & the id of btn w
        then
          delete item p of CWSList
          play harpsichord tempo 100 b#x7
          exit hitCBtn
        end if
      end repeat
      put "btn id " & the id of btn w & ","after CWSList
      play harpsichord tempo 100 b#x7
    end HitCBtn
```


© The Definitive MacTutor, Vol. 4

This section moves the objects.

```
on movegroup
  global CWSList
  wait until the mouse is down
  put the mouseLoc into mLoc
  put empty into mH
  put empty into mV
  repeat with i = 1 to the number of items in CWSList
    put item i of CWSList into iList
   put item 1 of the loc of iList - item 1 of mLoc & "," after
mH
    put item 2 of the loc of iList - item 2 of mLoc & "," after
mV
  end repeat
  repeat while the mouse is down
    get the mouseLoc
    lock screen
    repeat with i = 1 to the number of items in CWSList
      put it into iList
      add item i of mH to item 1 of iList
      add item i of mV to item 2 of iList
      set loc of item i of CWSList to iList
    end repeat
    unlock screen
  end repeat
end movegroup
```

I added this section very quickly because I already had the underlying selection and moving structures. As you can see it is very simple.

```
on arrowkey whichkey
  global CWSList
  if the mouse is down then
    if whichkey = "left" then
      put the mouseH into L
      repeat with i = 1 to the number of items in CWSList
        put the width of item i of Cwslist into w
        set the left of item i of Cwslist to L
        set the width of item i of Cwslist to w
      end repeat
    end if

    if whichkey = "right" then
      put the mouseH into R
      repeat with i = 1 to the number of items in CWSList
        put the width of item i of Cwslist into w
        set the right of item i of Cwslist to R
        set the width of item i of Cwslist to w
      end repeat
    end if

    if whichkey = "up" then
      put the mouseV into T
      repeat with i = 1 to the number of items in CWSList
        put the height of item i of Cwslist into h
        set the top of item i of Cwslist to T
        set the height of item i of Cwslist to h
      end repeat
    end if

    if whichkey = "down" then
      put the MouseV into B
      repeat with i = 1 to the number of items in CWSList
        put the height of item i of Cwslist into h
        set the bottom of item i of Cwslist to B
        set the height of item i of Cwslist to h
      end repeat
    end if
    exit arrowkey
  end if
  pass arrowkey
end arrowkey --**AutoHyperEdit**--
```

Non-Comercial use only - All Rights Reserved
Fred Stauder, Hypercard Editor, (HyperChat) MacTutor
---- AutoHyperEdit by Fred Stauder, June 1988 ---

You should get the stack because of the animation and sound ideas it contains. The key thing is to plan your stacks keep them modular so that they can be added to and be polite and use the *pass* statement so that other peoples scripts will work. For example a stack that uses *on mouseStillDown* and does not *pass* it will stop AutoHyperEdit working in that Stack.

Send your Articles, ScriptTips, SpeedTips, etc. to:

Fred Stauder, AppleLink D0280
Ecofin Reseach and Consulting
Lavaterstrasse 45
Zurich 8002
Switzerland
or
To the MacTutor Office

HyperChat™
XCMD Cookbook: Sorting Routines

Donald Koscheka
Apple Computer, Inc.
MacTutor Vol. 4 No. 8

In the last two issues we covered XCMD programming basics. First we looked at the interface between Hypercard and XCMDs and then we discussed the callbacks and how they can make your XCMD programming a little easier. This month, I present an XCMD that adds a useful feature to Hypercard - the ability to sort lines in a text field.

This is a good example of XCMD programming since it extends Hypercard by adding a new feature. In addition to adding features, XCMDs are a useful way of speeding up some process that may have been written in HyperTalk. Sorting lines of text is certainly something that could be accomplished in Hypertalk. But if you have a lot of lines in the field, the Hypertalk script may be too slow. Enter the XCMD!

Before we can write the sort program, we must decide what it is we want to sort and what form the data will be presented in. Let's define a line as a string of text that is spearated by a newline character. Next we'll want to be able to sort both alphabetic strings and numeric strings. We need the numeric sort because strings of characters are sorted by ASCII value rather than numeric value.

Once we decide on the form and value of the data to sort, we can decide what we need to pass to the XCMD from Hypercard. Obviously, we'll need the lines. Parameter 1 will contain a handle to the field, parameter 2 will tell the XCMD whether to sort by ASCII value (alphanumeric sort) or by numeric value (decimal sort). Parameter 3 tells the xcmd to sort in ascending or descending order. Actually LineSort only sorts in ascending (smallest to largest) order. There really is no need to sort from largest to smallest. You can simply report the lines "backwards" to get descending order. I've left this as an exercise to the reader since it involves little more than reversing the order of the FOR loop in two of the routines in SetNewText and SetNewNums.

When LineSort is complete, we'll return the sorted list in the parameter block's returnValue. By making this an XFCN we can invoke it using the follwing format:

```
Put LineSort( card field "my list", alpha ) into card field
"my list"
```

The actual process of sorting can be broken down into the following steps:

(1) LineStart: Create an array of offsets into the list marking the start of each line.
(2) SortText: Sort the list of text by comparing lines using some sorting algorithm.
(3) SetNewText: Report the sorted list back to Hypercard.

(Note: the process is identical for numeric sorting. Because the numeric sort is quite a bit simpler in implementation, we'll concentrate this discussion on the text sort).

How we accomplish these three tasks is quite another story. Sorting lines of text is not as straightforward as sorting integers or characters since each line can be of an arbitrary length. Before performing the sort, we need to calculate where in the large chunk of text each line starts. We create an array, LineStart, that gives us an offset into the list of the start and length of each line. Consider the list:

Margaret
Colleen
Donald

The first line in the list has an offset of 0 and a length of 8. The next item in the list has an offset of 9 and a length of 7. The third item has an offset of 17 and a length of 6. If the numbers don't seem to add up it's because the linestart array accounts for the newline character that terminates every entry in the list!

This matter of creating a linestart array pays handsome dividends. When it comes time to sort the list, we don't have to move the strings around in memory. Rather, we simply rearrange the linestart array to reflect the sort order. For example, the line start array for the above array looks like this before the sort:

[0,8]
[9,7]
[17,6]
and like this after the sort:
[9,7]
[17,6]
[0,8]

To report the sorted list back to hypercard now becomes a simple task. SetNewText extracts N elements from the linestart array where N is the number of elements in the array. We determine this number by dividing the size of the array by the size of one element in the array. For each element in the array, the first value in the array is the offset of the first byte in the string and the second element is the number of bytes to copy onto that line. Each line is already terminated with a newline so the process of building the output string is a simple process of concatenating the strings in the order prescribed by LineStart. If you want to perform a descending sort, build the output list starting with the last element in the array and work your way backwards, one element at a time! This is good practice for the beginner so I haven't included the sort order part of the code.

© The Definitive MacTutor, Vol. 4

I glossed over the actual sorting of the linestart array for two reasons. First off, it looks a lot scarier than it is. Second off, you may want to replace my sorting algorithm with one of your own. I use a Shell sort because it is one of the easier sorting methods to follow. If you're programming in MPW "C" you should consider replacing SortText and SortNums with Apple's QuickerSort. Greg Kimberly of Apple Computer, Inc. demonstrated an XCMD to me that uses QuickerSort. It was at least three times faster than my Shell sort implementation!

The Shell sort starts by comparing the first element in the array with every element that is at least *jump* elements away from the first element. If an some element is smaller than the first element, then we swap that element, and test the next one. Once no more elements can be swapped, we exit the FOR loop, decrease the jump size and start the comparison process all over again.

The ToolBox call IUMagString compares the two string elements and returns -1 if string 1 is less than string 2, 0 if they're equal and 1 if string 1 is greater (the swap condition). IUMagString takes two string pointers and the length of each string as input. The length is easy, that's simply the second element of each linestart array entry. We can create a pointer to each line by adding the offset element of each array entry to the starting address of the text. Since the text is referenced via a handle (passed to sortText as hField), we lock down the handle to make sure that this starting address doesn't change on us during the sort.

The numeric sorting scheme is very similar to the alpha scheme, only easier because the size of each element is fixed at 4 bytes (the size of a longInt). If you have trouble reading the sort Text routines, try starting with the sort numeric code.

Next month: File I/O. You can take it with you!

```
{*************************}
{* File: LineSort.p                    *}
{*                                     *}
{* Sorts lines of text (delimited by   *}
{* newline and null terminated).       *}
{* Returns the sorted  container.      *}
{*-------------------                  *}
{* In:   paramPtr=pointer to the XCMD  *}
{*       Parameter Block               *}
{*                                     *}
{*   params[1] = handle to the text    *}
{*   params[2] = handle to sort type   *}
{*       (ALPHA, NUMERIC)              *}
{*       params[3] = Sort Order        *}
{*       (ASCENDING, DESCENDING)       *}
{*                                     *}
{* Defaults : ALPHA, ASCENDING         *}
{* Out:  returnValue = h andle to the  *}
{*           sorted data               *}
{*                                     *}
{*-------------------                  *}
{* © 1988, Donald Koscheka             *}
{*       All Rights Reserved           *}
{*-------------------                  *}
{*************************}

{*************************
        BUILD SEQUENCE
```

```
pascal LineSort.p
link -m ENTRYPOINT -rt XFCN=65535 ∂
     -sn Main=LineSort ∂
     LineSort.p.o ∂
     "(Libraries)"Interface.o ∂
     "(PLibraries)"Paslib.o ∂
     -o "(xcmds)"testxcmds

*************************

{$S LineSort }

UNIT Donald_Koscheka;

{———INTERFACE———}

INTERFACE

USES
     MemTypes,QuickDraw,OSIntf,ToolIntf,
     PackIntf,HyperXCmd;


PROCEDURE EntryPoint(paramPtr:XCmdPtr);

{———IMPLEMENTATION———}

IMPLEMENTATION

{$R-}
CONST
     NULL          = $00;
     NEWLINE       = $0D;
     ALPHA            = $00;
     NUMERIC       = $01;
     ASCEND        = $00;
     DESCEND       = $01;

     LESSTHAN      = -1;
     EQUALTO       = 0;
     GREATERTHAN   = 1;

TYPE
     Str31         = String[31];

     LinePtr  = ^LineElem;
     LineHand = ^LinePtr;
     LineElem = PACKED RECORD
          Start    : LongInt;
          Size : LongInt;
     END;

     numPtr   = ^LongInt;
     numHand  = ^numPtr;


PROCEDURE LineSort(paramPtr:XCmdPtr);
                                   FORWARD;

{———EntryPoint———}

PROCEDURE EntryPoint(paramPtr: XCmdPtr);
     BEGIN
          LineSort(paramPtr);
     END;

{———LineSort————}

PROCEDURE LineSort(paramPtr:XCmdPtr);
(*************************
* Main code segment  follows
*************************)
VAR
```

```
    SortType      : INTEGER;
    SortOrder     : INTEGER;
    LineStart     : LineHand;
    hNums         : numHand;
    NewField      : Handle;
    SortStr       : Str255;

{$I XCmdGlue.inc }


(************************)
(*** Alpha Sorting Routines     ***)
(************************)


FUNCTION GetLineStarts(hField:Handle)
                  : LineHand;
(*************************
* Given a pointer to a block of text,
* scan for line-terminators (NEWLINE
* | NULL) and  fill out a dynamically
* allocated array of line starts
* information.
*
* The line starts array contains
* the offset in the text to the start
* of the line as well  as the length
* of the line in bytes.
*
* Offsets are used to allow the record
* to remain valid across relocation
* of the basic text.
*
* In: Pointer to a block of text,
* null terminated.
*
* Out: Handle to an array of
* linestart indices.
*
*************************)
VAR
    done     : BOOLEAN;
    arraySize: LongInt;
    startText: LongInt;    { Base pointer of the text }
    sizeText : LongInt;    { length of the current run}
    lineStart: LineHand;   { array of lineStarts pointers}
    txStart  : Ptr;        { pointer to the input text}
    txPtr    : Ptr;        { pointer into input text (FIELD)}
    lineArray: LinePtr;    { Pointer into lineStart array }

BEGIN
    IF (hField<>NIL)AND(hField^^<>NULL)THEN
    BEGIN
    HLock( hField );
    txPtr    := hField^;
    startText := ORD( txPtr );
    lineStart := LineHand(NewHandle(0));
    done := FALSE;

    WHILE NOT done DO
    BEGIN
        txStart := txPtr;
        ScanToReturn( txPtr );

        IF txPtr^ = NULL THEN
        BEGIN
            txPtr^   := NEWLINE;
            done := TRUE;
        END;

        txPtr := Pointer( ORD(txPtr) + 1);
        sizeText := ORD(txPtr)-ORD(txStart);

        IF sizeText > 1 THEN
        BEGIN
        (*** point to next record in linestarts array ***)
        arraySize := GetHandleSize( Handle(lineStart) );
        SetHandleSize( Handle(lineStart),
                    arraySize + sizeOf( lineElem ));
        lineArray := Pointer( ORD(lineStart^) + arraySize );

        (*** Put away offset and length of line ***)
        WITH lineArray^ DO
            BEGIN
                Start:= ORD(txStart) - startText;
                    Size := SizeText;
            END;
        END;
    END; (*** While ***)

    HUnlock( hField );
    END;
    GetLineStarts := lineStart;
END;


PROCEDURE  SortText( hField:Handle );
(*************************
* Given a handle to an run of text,
* sort the lines of text pointed to and
* rearrange the array accordingly.
*
* Text is sorted by rearranging the line
* starts array.
*
* In:    Pointer to  array of linestarts
*        linestarts, linecount (global)
* Out:   sorted array of linestarts.
*************************)
VAR
    done         : BOOLEAN;
    jump,len1,len2: INTEGER;
    n,m,lineCount: LongInt;
    str1, str2   : Ptr;
    tempElem     : LineElem;
    elem1, elem2 : LinePtr;

BEGIN
    LineCount    := GetHandleSize( Handle(lineStart) );
    LineCount    := LineCount DIV sizeOf( LineElem );

    jump := lineCount;
    HLock( Handle(lineStart) );
    HLock( hField );

    WHILE jump > 1 DO
    BEGIN
        jump := jump DIV 2;
        REPEAT
            done := TRUE;
            FOR m := 0 to ( lineCount - jump - 1 ) DO
            BEGIN
            n := m + jump;

    (*** Calculate the offsets of the two elements ***)
            elem1 := LinePtr( ORD(LineStart^) +
                    (n * sizeof( LineElem )) );

            str1 := Pointer( ORD( hField^ ) + elem1^.Start) ;
            len1:= INTEGER( elem1^.Size );

            elem2    := LinePtr( ORD(LineStart^)  +
                     (m * sizeof( LineElem )) );

            str2 := Pointer( ORD( hField^ ) +
                    elem2^.Start) ;
```


© The Definitive MacTutor, Vol. 4

```
            len2 := INTEGER( elem2^.Size );

            IF IUMagString( str2, str1, len2, len1 )
            = GREATERTHAN THEN
            BEGIN
                tempElem.Start    := elem1^.Start;
                tempElem.Size := elem1^.Size;
                elem1^.Start  := elem2^.Start;
                elem1^.Size       := elem2^.Size;
                elem2^.Start  := tempElem.Start;
                elem2^.Size       := tempElem.Size;
                done := FALSE;
            END;
        END; (*** FOR loop ***)
        UNTIL done;
    END; (*** WHILE jump > 1 ***)

    HUnlock( Handle(lineStart) );
    HUnlock( hField );
END;


FUNCTION SetNewText( hField: Handle;
                     Dir : INTEGER ): Handle;
(****************************
* Given a pointer to a linestarts array, and
* a corresponding block of text, rearrange
* the text to match the line starts in the
* array.  This is useful after doing a search
* since the linestarts array will be in order
* but the text won't be.
*
* return a run of null-terminated text
* with each line NEWLINE-terminated.
*
* In:    Handle to text array
*        Handle to linestarts array.
*
* Out: Handle lines of newline terminated text.
****************************)
VAR
    LineNum,
    LineCount,
    oldSize          : LongInt;
    NewText          : Handle;
    StartOfLine      : Ptr;
    NextLine         : LinePtr;

BEGIN
    LineCount     := GetHandleSize( Handle(lineStart) );
    LineCount     := LineCount DIV sizeOf( LineElem );

    NewText       := NewHandle( 0 );

    FOR LineNum := 0 to LineCount-1 DO
        BEGIN
            NextLine := LinePtr( ORD( LineStart^ ) +
             (LineNum * sizeOf( LineElem )) );
            StartOfLine := Pointer( ORD( hField^ ) +
             NextLine^.Start );

(*** add length of new line to output text ***)
        oldSize := GetHandleSize( NewText );
        SetHandleSize( NewText, oldSize + NextLine^.Size );

    (*** move the line into the array ***)
            BlockMove( StartOfLine, Pointer( ORD( NewText^ )
+ oldSize), NextLine^.Size );
        END;

    (*** Tack a NULL on the end of the new text ***)
    oldSize  := GetHandleSize( newText );
    SetHandleSize( newText, oldSize + 1 );
    StartOfLine  := Pointer( ORD( newText^ ) + oldSize );
    StartOfLine^ := NULL;
    SetNewText   := NewText;
END;



(****************************)
(*** Number Sorting Routines    ***)
(****************************)


FUNCTION GetNums( hField : Handle ): numHand;
(****************************
* Given a handle to a block of text, scan
* for line-terminators (NEWLINE | NULL) and
* fill out a dynamically allocated array of
* long integers, one for each line.
*
* Out: Handle to an array of longInt.
*
****************************)
VAR
    done     : BOOLEAN;
    arraySize,
    theNum   : LongInt;
    txStart  : Ptr;
    txPtr        : Ptr;
    numArray : numPtr;
    hNum     : NumHand;
    tStr     : Str255;

BEGIN
    IF ( hField <> NIL ) AND ( hField^^ <> NULL ) THEN
    BEGIN
        HLock( hField );
        txPtr    := hField^;
        hNum := NumHand( NewHandle( 0 ) );
        done := FALSE;

        WHILE NOT done DO
        BEGIN
            txStart := txPtr;
            ScanToReturn( txPtr );

            IF txPtr^ = NULL THEN
                done := TRUE
            ELSE
                txPtr^ := NULL;

            ZeroToPas( txStart, tStr );
            theNum   := StrToNum( tStr );
            txPtr    := Pointer( ORD(txPtr) + 1);

            arraySize := GetHandleSize( Handle( hNum ) );
            SetHandleSize( Handle( hNum ), arraySize +
sizeOf( longInt ));
            numArray := Pointer( ORD( hNum^) + arraySize );
            numArray^:= theNum;

        END; (*** While ***)
        HUnlock( hField );
    END;
    GetNums := hNum;
END;

PROCEDURE SortNums( theNums : NumHand );
(********************
* Given a pointer to an array of longints,
* sort the numbers  and rearrange the array
* accordingly.
*
* In:    Handle to an array of longInts;
```

```
*       uses linecount and lineNum;
* Out:   sorted array of linestarts.
*********************)
VAR
    done              : BOOLEAN;
    jump,len1,len2    : INTEGER;
    n,m  ,swap,
    lineCount         : LongInt;
    num1,num2         : numPtr;

BEGIN
    LineCount := GetHandleSize( Handle(theNums) ) DIV sizeOf(
LongInt );
    jump := lineCount;

    WHILE jump > 1 DO
    BEGIN
        jump := jump DIV 2;
        REPEAT
            done := TRUE;
            FOR m := 0 to ( lineCount - jump - 1 ) DO
            BEGIN
                n := m + jump;

        (*** Calculate the offsets of the two elements ***)
                num1 := Pointer(ORD(theNums^) + (n * sizeof(
LongInt )) );
                num2 := Pointer(ORD(theNums^) + (m * sizeof(
LongInt )) );

                IF num2^ > num1^ THEN
                BEGIN
                    swap := num1^;
                    num1^:= num2^;
                    num2^:= swap;
                    done := FALSE;
                END;
            END; (*** FOR loop ***)
        UNTIL done;
    END; (*** WHILE jump > 1 ***)
END;


FUNCTION SetNewNums( hNum: numHand; Dir : INTEGER ): Handle;
(*********************
* Given a handle to an array of
* longints, convert  them back to
* strings and put them away in
*  a new handle as text
*
* In:    Handle to num array
*
* Out: Handle lines of newline
* terminated text.
*********************)
VAR
    index, intCount,
    oldSize, numLen   : LongInt;
    nexNum            : NumPtr;
    NewText           : Handle;
    tempPtr           : Ptr;
    tempStr           : Str255;

BEGIN
    NewText       := NewHandle( 0 );
    intCount := (GetHandleSize( Handle(hNum) ) DIV sizeOf(
LongInt ));

    FOR index := 0 to intCount-1  DO
        BEGIN
            nexNum    := NumPtr( ORD( hNum^ ) +
                         (index * sizeOf( LongInt )) );
            tempStr := NumToStr( nexNum^ );
            numLen  := LongInt( ORD( tempStr[0] ) );

        (*** add length of new line to output text  ***)
            oldSize := GetHandleSize( NewText );
            SetHandleSize( NewText, oldSize + NumLen + 1 );

        (*** move the line into the array           ***)
            TempPtr := Pointer( ORD( @tempStr ) + 1 );
            BlockMove( TempPtr , Pointer(ORD( NewText^)+
oldSize),NumLen);
            TempPtr := Pointer(ORD( NewText^) + oldSize +
NumLen);
            TempPtr^ := NEWLINE;
        END;

    (*** Tack a NULL on the end of the new text ***)
    oldSize       := GetHandleSize( newText );
    SetHandleSize( newText, oldSize + 1 );
    tempPtr       := Pointer( ORD( newText^ ) + oldSize );
    TempPtr^      := NULL;
    SetNewNums        := NewText;
END;


(******* Main Block for LineSort *******)
BEGIN
    NewField := NIL;

    WITH paramPtr^ DO
        IF (params[1] <> NIL) AND (params[1]^^ <> NULL) THEN
            BEGIN
                SortType := ALPHA;
                SortOrder := ASCEND;

                IF params[2] <> NIL THEN
                    BEGIN
                        ZeroToPas( params[2]^, sortStr );
                        IF StringEqual('NUMERIC', sortStr)
THEN
                            SortType := NUMERIC;
                    END;

                IF params[3] <> NIL THEN
                    BEGIN
                        ZeroToPas( params[3]^, sortStr );
                        IF StringEqual('DESCENDING',
sortStr ) THEN
                            SortOrder := DESCEND;
                    END;

                CASE SortType OF
                ALPHA:
                BEGIN
                    LineStart := GetLineStarts( params[1] );
                    SortText( params[1] );
                    newField := SetNewText( params[1],
SortOrder );
                    DisposHandle( Handle( LineStart ) );
                END;

                NUMERIC:
                BEGIN
                hNums := GetNums( params[1] );
                SortNums( hNums );
                newField := SetNewNums( hNums, SortOrder );
                DisposHandle( Handle( hNums ) );
                END;
                END; (*** CASE SortOrder OF ***)
            END;

    paramPtr^.returnValue := newField;
END;
END.
```

HyperChat™

Script Tips

Fred Stauder
HyperChat Editor
Zurich, Switzerland
MacTutor Vol. 4 No. 9

on HyperChat

— HyperEditorial

When to HyperTalk

People say that Hypercard is not a real programing language. Well what is a program? A program is a series of instructions that are processed by a computer to yield a result. We have established that Hypercard is a programing language; now people say it is not powerful. What is meant by a powerful programing language? Does it mean flexible, or easy to use, or even the fastest way to get a result?

The most flexible language is of course binary because that is the native language of the computer. I don't think anyone would call binary a powerful language, it would take too long to do anything really useful with it. Hypercard can do some graphic manipulations easier than Pascal or C. There are many things that can be done in Pascal and C that are too cumbersome in Hypertalk or even impossible. In the past I have shown things that have been thought to be impossible in Hypertalk. The effort was greater than doing them with external commands, but it could be done.

I believe we are entering an Era of what I call "Russian Doll" programing environments. A Russian Doll is a large wooden doll which you can open up to find a smaller doll. This process goes on many times. The analogy serves programing well. The first level is very simple we can, for instance, change command key equivalents (like QuickKeys). The next level we can move objects around (like buttons in Hypercard); the next level we can program the objects with scripts. We then get to the next level where we can add external code (C or Pascal). We should also be able to change the properties of the level above i.e. Hypercard objects. Unfortunately we can't. We are however at the point where we might soon be able to use one script or piece of code in different applications.

Now I come to my definition of a powerful programing environment: "A programing environment is powerful if it lets you have the flexibility to communicate to other environments and utilize routines that have been developed for other purposes." This is why MPW is regarded as a powerful programing tool. The developer still has to choose when it is best to use Assembly or Pascal or C. If the developer does not have the experience necessary in Assembly he can get some other person to do it. A similar thing happens in Hypertalk with external commands. Wouldn't it be nice to use some of your great scripts and buttons you have developed in Hypercard, in MacWrite?

This month we look at the similarities and differences between Hypertalk and Pascal in an article by Paul Merrill. Don Koscheka in XCMD Cookbook shows you an example when it is best to use Hypertalk and when to use XCMD'S.

—ScriptTips

Last month I brought you AutoHyperEdit the response has been great and I am glad to hear that it is finding uses especially for the handicapped. Using AutoScriptEdit a person can edit scripts quickly one handed. This month I had a request how to make buttons 1 or two pixels high. That was easy, you just type in the command *set the height of btn x to 2*. I wanted an easy way, so I added the following italic text to my arrowkey handler in AutoHyperEdit. So all you have to do is to shift select the button or field, then double click outside an object to get into the action mode and hold down the shiftkey and use the arrow keys to make the button or field smaller or larger. I could have made it act on groups of objects but I wanted to make it modular. I leave it to you, to make it work on groups, as an exercise.

```
on arrowkey whichkey-**AutoHyperEdit**-
  global CWSList
  if the shiftkey is down then
  put item 1 of cwslist into T
   put the width of T into TW
  put the height of T into TH
  if whichkey is "down" then set the height of  T to TH - 1
  if whichkey is "up" then set the height of  T to TH + 1
  if whichkey is "left" then set the width of  T to TW - 1
  if whichkey is "right" then set the width of  T to TW + 1
  end if
  if the mouse is down then
    if whichkey = "left" then
      put the mouseH into L
      repeat with i = 1 to the number of items in CWSList
        put the width of item i of Cwslist into w
        set the left of item i of Cwslist to L
        set the width of item i of Cwslist to w
      end repeat
    end if

    if whichkey = "right" then
      put the mouseH into R
      repeat with i = 1 to the number of items in CWSList
        put the width of item i of Cwslist into w
        set the right of item i of Cwslist to R
        set the width of item i of Cwslist to w
      end repeat
    end if

    if whichkey = "up" then
      put the mouseV into T
      repeat with i = 1 to the number of items in CWSList
        put the height of item i of Cwslist into h
        set the top of item i of Cwslist to T
        set the height of item i of Cwslist to h
      end repeat
    end if

    if whichkey = "down" then
      put the MouseV into B
      repeat with i = 1 to the number of items in CWSList
        put the height of item i of Cwslist into h
        set the bottom of item i of Cwslist to B
```

© The Definitive MacTutor, Vol. 4

```
        set the height of item i of Cwslist to h
      end repeat
    end if
    exit arrowkey
  end if
  pass arrowkey
end arrowkey --**AutoHyperEdit**-
```

The second script tip deals with searching in large numbers of cards for a checked item. If you have to check for the hilite of a button in 2000 cards, it takes a while. If you use a field with a check mark and search for the character in the field it is much, much faster because it uses Hypercards fast search algorithms. Simply lock the field and put this script into it. You could also use an "X" mark.

```
on mouseup
  if me is empty
  then put "✔" into me
  else put empty into me
end mouseUp
on HyperChat
```

© The Definitive MacTutor, Vol. 4

HyperChat™

Comparing HyperTalk to Pascal

Paul F. Merrill
Brigham Young University
Provo, Utah
MacTutor Vol. 4 No. 9

Pascal and HyperTalk: A Comparison

[Paul F. Merrill is a professor of instructional science and computer based education at BYU. He teaches courses in Pascal and computer based authoring systems. He has also written a book entitled, Computers in Education published by Prentice-Hall]

Introduction

HyperCard has produced a lot of excitement and discussion in the Macintosh community. It has generated a large following and no small number of critics. Many proponents have argued that HyperCard enables the weekend programmer to create sophisticated applications which tap the power of the Macintosh user interface without having to digest the formidable tool box and "Inside Macintosh." John Sculley has predicted that it will have an impact in the Macintosh world similar to the impact of Applesoft Basic in the Apple II world.

But just how powerful is the HyperCard authoring environment and its associated programming language, HyperTalk? One way to address that question is to compare HyperTalk with some standard language. Therefore, the purpose of this article is to compare HyperTalk with the Pascal programming language. This comparison will not only reveal the power of HyperTalk, but it will also provide a bridge for the Pascal programmer to cross over into the new exciting world of HyperCard.

This article is neither a tutorial in Pascal or HyperTalk. It is assumed that the reader has some familiarity with Pascal and has at least seen a demonstration of or read a good review article on HyperCard.

The following comparisons will be done by annotated examples rather than by complicated syntax diagrams. Since one or two examples cannot show the full range of possibilities for a given command, page number references are provided to a full discussion of the command found in Danny Goodman's excellent book, "The Complete HyperCard Handbook." Pascal examples are shown in boldface while HyperTalk examples are shown in italics type. Annotations or comments are shown within curly brackets. Single spacing will be used to indicate commands that go together as a set, while double spacing will be used to separate independent examples.

Program structure

Unlike Pascal, HyperTalk cannot be used to create stand-alone programs. It can only be used within HyperCard to control the activity of HyperCard objects (stacks, backgrounds, cards, fields and buttons). HyperTalk is not used to create a "program" that is executed from beginning to end. Instead each HyperCard object may have associated with it HyperTalk commands called a "script". Each script may contain several Pascal like procedures called "message handlers." The commands within a message handler are executed when an object receives a message which matches the handler. HyperCard sends messages to an appropriate object whenever certain events occur. For example, a button script may have a message handler which will instruct HyperCard to go to the next card, using a dissolve visual effect, when the mouse is clicked on the button and the "mouseUp" message is received. The script associated with that next card might contain a message handler which will be executed when the card is opened and a corresponding "openCard" message is received by the card (see Goodman, pgs. 345-360). An example of a simple Pascal program and two HyperTalk message handlers are shown below. The Pascal program calls a procedure "getname" which requests the user's name and then displays "Thank you" on the screen. The first HyperTalk message handler performs essentially the same functions. It is executed when a specific card is opened. The second message handler is executed when the mouse is clicked on a button. HyperTalk message handlers begin with the word "on" followed by the name of a particular message. They always terminate with the word "end" followed by the same message name. In these and subsequent examples, you will find that most HyperTalk commands are closer to standard English commands than their Pascal counterparts.

```
program example;
        procedure getname;
          var
            name: string;
          begin
            writeln('What is your name?');
            readln(name);
            writeln('Thank you')
          end;
        begin
          getname
        end.

on openCard
        ask "What is your name?"
        put "Thank you" into field 1
end openCard

on mouseUp
        visual effect dissolve
        go to next card
end mouseUp
```

Simple variables & assignment statements

In Pascal, the programmer has to declare what type of values will be stored in a variable before it can be used. Local variables do not need to be declared in HyperTalk. Global variables are

declared in any handler which uses them. All HyperTalk variables are of type text. Numeric characters are automatically converted to numbers before mathematical operations are performed (Goodman, p. 362). In addition to variables, information may be stored and retrieved from other "containers" such as fields, the Message Box, and any text selected by the user (p.400). The following Pascal examples show the declaration of several different types of variables followed by three assignment statements where mathematical expressions are evaluated and variables are given a value. The HyperTalk "put" command also evaluates expressions and stores values into variables or other containers. Note the additional HyperTalk arithmetic commands.

```
var
  x, y, sum,counter: integer;
  name: string;
  r: real;

name:="Paul";

sum:= (x+y)*r/2;

counter:=counter +1;
```

```
put "Paul" into name          (p. 415)
put (x+y)*r/2 into sum        (p. 415)
add 1 to counter              (p. 441)
subtract  field 1 from sum    (p. 442)
multiply salary by selection  (p. 444)
divide field "total" by  num  (p. 445)
```

Simple Input/output

Macintosh Pascal provides a text and a drawing window for the display of text or graphics. Multiple windows can be displayed on the screen using tool box routines in LightSpeed Pascal. The present version of HyperCard only allows one window or card to be displayed on the screen at a time.*[In HC version 1.2 and later you can selectively hide and show Card and Background Picts giving you more graphic control. HyperEd]* However, HyperCard differentiates between text fields and a graphics layer. In the following examples the Pascal "write" command is used to display a simple string in the text window, while the HyperTalk "put" command is used to display information in a text field.

```
write ('Display on the text screen');

put "Display in field" into field 2        (p. 415)
```

These examples show how values from variables and literal strings are combined for display on the text screen or in a field. In HyperTalk, the double ampersand (&&) adds a space.

```
write (str1 ,' literal string ', str2,' ', str3);

put str1 & " text " & str2 && str3 into field  "Hello"
```

The results of the evaluation of a mathematical expression may also be mixed with literal strings. The ":7:2" after the expression is used to format the resulting real number in a seven column field with two digits after the decimal. HyperTalk requires an additional command, "set numberFormat," in order to format the real number.

```
write ('Your salary is ', rate*hours:7:2);

set numberFormat to "0.00"    (p. 489)
put "Your salary is " & rate*hours into field "salary"
```

The Pascal "writeln" command is exactly the same as the "write" command, shown above, except it places a "carriage return" character at the end of the line. To do the same thing in HyperTalk, the "return" character must be given explicitly as shown in the following example. However, in HyperTalk it is generally not necessary to specify the "return" character since the text will automatically be "word wrapped" within the dimensions of the field. This capability makes it possible to use one HyperTalk "put" command where several Pascal "writeln" commands would be necessary.

```
writeln ('Text followed by return');

put "Text" & return after field "display" (p.415)
```

Macintosh Pascal provides three commands, "drawChar," "drawString, " and "writeDraw" for displaying text on the drawing screen. Each of these must be preceded by a "moveto" command which specifies the coordinates of where the text is to be displayed. In HyperCard, the "type" command is used to place text in the graphics layer of a card. This command must be preceded by other commands which select the text tool and specify the location of where the text is to be placed. For many applications, text may be placed in fields and on the graphics layer of a card using the MacPaint like tools built into HyperCard rather than using HyperTalk commands. Text placed in fields may be edited by the user, while painted text in a graphics layer cannot be edited. However, painted text may be erased using the erasure tool or moved using the lasso tool.

```
moveto (40,100);
writeDraw ('Display in drawing window');

choose text tool    (p. 429)
click at 40,100     (p. 430)
type "This is painted text"   (p. 433)
```

In Pascal, user input from the keyboard can be obtained by using the "read" or "readln" commands. These are generally preceded by a prompt displayed by the "write" or "writeln" commands. HyperTalk provides several techniques for obtaining keyboard input from the user. The "ask" command displays a dialog box with the first string argument as a prompt and the second string as the default selected response. The user's

© The Definitive MacTutor, Vol. 4

response is stored in the HyperTalk system variable called "it." The "answer" command displays a dialog box with the first string argument as a prompt and two or three pushbuttons each labeled by the succeeding strings. When the user clicks a button, the button label is placed in the variable "it." Unfortunately, the location of these dialog boxes cannot be controlled by HyperTalk. The "put" command may be used to place a copy of whatever the user types into an editable field into a specified variable, while the "get" command places the contents of a field in the system variable "it."

```
write ('Please enter your name ');
readln(name);

ask "Please enter your name" with "Paul"  (p. 465)

answer "Are you happy" with "yes" or "no" or
                    "maybe"                (p. 463)

put field 1 into address (p. 415)

get field name            (p. 419)
```

Operators and functions

The following math and boolean operators are the same in both languages. However, HyperTalk includes the additional operators: "is" and " is not" (p. 563-576).

+ - * / div mod = <> < <= >= and or not

The Pascal "random" function returns a random number between -32767 and +32767, while "the random" function in HyperTalk returns a random number between 1 and a specified value.

```
x:=random;

put the random of upper into x            (p. 545)
```

The following math functions are the same in both languages. HyperTalk also provides several additional functions such as "annuity" and "average" (p. 546-551).

round, trunc, abs ln, sqrt, cos, sin, exp

Sequence control structures

Both Pascal and HyperTalk provide powerful if-then-else control structures with very similar syntax. The following examples display different messages on the text screen (Pascal) or in a field (HyperTalk) and increment different variables (count or ave) depending on the values of the variables "status" and "gpa." There is no HyperTalk equivalent of the Pascal "case" statement other than nested "if-then-else" commands (p. 589).

```
if (status = 'undergrad') and (gpa > 3.7) then
    writeln('Deans list')
  else if gpa < 2.0 then
    begin
      writeln('Probation');
      count := count + 1
    end
  else
    begin
      writeln('Average');
      ave := ave + 1
    end;

if stat = "undergrad" and gpa > 3.7 then  (p. 584)
    put "Deans List" into field 1         (p. 415)
  else if gpa < 2.0 then                  (p. 587)
    put "Probation" into field 1          (p. 415)
    add 1 to count                        (p. 441)
  else                                              ( p .
587)
    put "Average" into field 1            (p. 415)
    put ave + 1 into ave                  (p. 415)
  end if
(p. 587)
```

Both Pascal and HyperTalk provide several very powerful repetitive control structures with similar syntax. However, HyperTalk provides an additional structure not found in Pascal and also adds several abnormal exit features not shown in the examples: "next repeat," "exit repeat," "exit if," and "exit <handler name> (p. 594). The following repeat loops will continue asking for a number from the user and display the value of the number multiplied by 10 until the user enters the number "0."

```
repeat
  Writeln('Enter a number');
  readln (it);
  writeln(it*10);
until it = 0;

repeat until it = 0       (p. 591)
  ask "Enter a number"    (p. 465)
  put it *10 & return after field 1       (p. 415)
end repeat
```

The following "while" loops will continue asking the user to enter a number and display the dividend produced by dividing the number into 50 until the user enters the number "0."

```
writeln ('Enter a number');
readln (it);
while it<>0 do
  begin
    writeln (50/it);
    writeln('Enter a number');
    readln(it);
  end;

ask "Enter a number"          (p. 465)
repeat while it is not 0      (p. 592)
  put 50/it  & return after field "answer"(p. 415)
  ask "Enter a number"        (p. 465)
end repeat
```

These "for do" loops will display the product of the integers 5 through 20 each multiplied by 5. The Pascal "for do" may also use "20 downto 5" while the HyperTalk "repeat with" may use

"20 down to 5."

```
for i:= 5 to 20 do
  writeln(i*5);

repeat with i = 5 to 20 (p. 593).)
  put i*5 & return after field "product"
end repeat
```

This HyperTalk loop will repeat 20 times. Each time it will place the string "I love you" into field 3 and sound a beep.

```
(No Pascal equivalent)

repeat 20 times          (p. 590)
  put "I love you " after field  3        (p. 415)
  beep 1                                  (p. 469)
end repeat
```

QuickDraw Graphics

Pascal and HyperTalk both provide commands for accessing the Macintosh QuickDraw Graphics routines. In most HyperCard stacks, the use of HyperTalk graphics commands within scripts will not be necessary since the drawing of graphics can be done with the MacPaint like tools included within HyperCard. In fact, HyperTalk graphics commands, used in scripts, actually provide access to the same tools found in the HyperCard tools menu. Therefore, the equivalent of most Pascal graphics commands are implemented in HyperTalk by selecting a particular graphics tool using the "choose" command and then simulating the dragging of the mouse across the screen using the "drag" command. Note that the arguments for the "drag" command are in a different order than those of the Pascal "frameRect" command. Ovals and rounded rectangles can also be drawn with HyperTalk by first selecting the corresponding tool. Three or four HyperTalk commands are required to implement the equivalent of some Pascal graphics commands such as "paintCircle" and "invertRect."

```
drawLine (x1,y1,x2,y2);

choose line tool    (p. 429)
drag from x1,y1 to x2,y2 (p. 431)

frameRect (top,left,bottom,right);

choose rectangle tool    (p. 429)
drag from left,top,right,bottom          (p. 431)

paintRect (r);

set filled to true  (p. 495)
choose rectangle tool    (p. 429)
drag from x1,y1 to x2,y2 (p. 431)

paintCircle (x1,y1,r);

set filled to true  (p. 495)
set centered to true     (p. 495)
choose oval tool    (p. 429)
drag from x1,y1 to x1,y2 (p. 431)

invertRect(r);

choose select tool  (p. 429)
drag from x1,y1 to x2,y2 (p. 431)
doMenu "invert"     (p. 421 & 247)

eraseRect (r);

choose select tool  (p. 429)
drag from x1,y1 to x2,y2 (p. 431)
doMenu "clear picture"   (p. 421)

penSize (4,4);

set lineSize to 4   (p. 497)

penPat (gray);

set pattern to 10   (p. 498)

penMode (patXor);

(No Hypertalk equivalent.)
```

Pascal provides several commands for controlling the properties of text displayed on the drawing screen by subsequent writeDraw commands. In HyperTalk, many properties of painted text, buttons, and fields may be changed using the "set" command.

```
textFont (3);

set textFont to Chicago (painted text (p. 499))

textFace ([bold,underline]);

set textStyle of button ok to bold, underline  (p. 499)

textSize(18);

set textSize of field 2 to 18 (p. 499)

textMode (srcXor);

(No HyperTalk equivalent .)
```

Mouse Control

The Macintosh tool-box ROM contains several routines, which can be accessed by Pascal commands, for tracking mouse events such as clicking the mouse button and moving the mouse. The "button" function returns a value of "true" if the mouse button is pressed down at the time the function is called; otherwise it returns "false." The "getMouse" procedure returns the coordinates of the position of the mouse pointer at the time the procedure is called. Although there are directly equivalent commands in HyperTalk, as shown below, mouse events are mainly tracked by message handlers which trap for mouse related messages: "mouseDown," "mouseUp," "mouseEnter," "mouseLeave," "mouseWithin," and "mouseStillDown" (p. 383-386). A "mouseUp" message handler is generally used to respond to normal mouse clicks within buttons, fields and card. In the examples below, "the mouseLoc" function returns the current position of the mouse pointer, while "the clickLoc" function returns the position of the mouse at the last click.

```
tf:=button;

put theMouseClick into tf     (p. 534)
```

```
getmouse(x,y);

put the mouseLoc into xy (p. 529)

put the clickloc into xy  (p.  535)
```

String procedures and functions

Macintosh Pascal provides several powerful commands for manipulating strings. HyperTalk provides even greater power by making it possible to manipulate characters, words, lines and items within any container.

```
it:=length (str);

get the length of word 3 of field "input" (p. 537)

it:=concat(str1,' literal ', str2,' ', str3);

put str1 & " literal string " & str2 && str3 into it
                                        (&& adds a
space (p. 415))

it:=copy(source, firstchar, numofchar);

get char 3 to 8 of line 3 of field "source"    (p. 419)

delete (source, firstchar, numofchar);

delete item 3 to 8 of field 3 (p. 420)

insert(source, dest, 5);

put selection before  fifth word of field 2    (p. 415)
```

The Pascal command "pos" is used to find the location of a pattern in a source string. It returns a number indicating the location of the first occurrence of the pattern. Zero is returned if the pattern is not found. The HyperTalk "offset" command performs exactly the same function. Two other HyperTalk commands, "contains" and "is in" return the value "true" if the pattern is found in the source.

```
if pos (pattern, source) <> 0 then
  writeln ('Correct');

if offset (pattern, source) <> 0 then   (p. 538)
  put "correct" into field "feedback"   (p. 415)

if source contains pattern then         (p. 573)
  put "correct" into field "feedback"   (p. 415)

if  pattern is in source then (p. 573)
  put "correct" into field "feedback"   (p. 415)
```

Sound Commands

A single voice note can be played by using the "note" command in Pascal. This command requires three integer arguments which specify the pitch, loudness and duration of one note. The HyperTalk "play" command is more versatile with arguments to indicate the instrument or waveform, the overall tempo, and a series of notes specified by normal musical symbols. For example, the notation "c#4e." would indicate a dotted-eighth c-sharp note in the 4th octave (middle octave). The "play" command can also be used to produce various self contained digitized sound effects. Such sounds may be recorded using a digitizer and converted to a Mac resource file.

```
note (440, 75, 48)

play "harpsichord" tempo 380 "b3 c#4 d e f# g a#3"

play "breaking glass"    (p. 470)
```

User defined types: arrays and records

On first glance, HyperTalk doesn't seem to support user defined types such as arrays and records. However, a card is analogous to a record which can contain many different named fields. Each field can contain up to 32 K. Field data can be accessed by line, item, word or character. Lines are separated by carriage returns, items by commas, and words by spaces. Data fields may be hidden from view if desired. The following HyperTalk script will compute the average of a set of numbers found in a field labeled "scores" for each card in a stack. Each number is separated by a space and considered to be a word.

```
Type scorearray = array [1..25] of integer;
  student = record of
      name: string[35];
      data: scorearray;
    end;
  class = array [1..100] of student;

repeat with n = 1 to number of cards      {p. 593}
  go to card n                            {p. 407}
  put 0 into sum                          {p. 415}
  put the number of words in field "scores" into
    num                                   {p. 415}
  repeat with i = 1 to num                {p. 593}
    add word i of field "scores" to sum   {p. 441}
  end repeat
  put sum/num into field average          {p. 415}
end repeat
```

Disk file input/output

Pascal has three different commands to open a file. "Rewrite" is used to open a new sequential file, "reset" is used to open a sequential file that has been previously created, and "open" is used to open a new or previously created random access file. These commands can be used to open simple text files or structured or typed files. In contrast, HyperTalk only deals with simple text files and only has one open command:

```
rewrite (f,'filename');

open file "filename"    (p. 475)
reset (f,'filename');

open (f,'filename');
```

In HyperTalk, if several fields are written to a file they should be separated by writing a "tab" character to the disk (write tab to file "filename"). Records or cards should be separated by writing a Return character to the file (write return to file "filename").

```
write (f, class[i].data);

write field "data" of card i to file "filename"  (p. 478)
```

The HyperTalk "read" command just reads characters from the file until it finds a delimiter such as "tab" or "return." The text is placed in the variable "it". The delimiter characters should be removed before placing the text in a field.

```
read(f,class[i].data);

read from file "filename" until tab        (p. 476)

close(f);

close file "filename"    (p. 475)
```

User defined functions and subroutines or procedures

The specification and calling of user defined functions in Pascal and HyperTalk is almost identical. The following example functions require two integer parameters and return the value of the first parameter raised to the power specified by the second parameter.

```
function power (num, exp: integer): integer;
  var i, answ: integer;
  begin
    answ:=1;
    for i:=1 to exp do
                          answ:=answ*num;
    power:=answ;
  end;

function power num,exp          (p. 561)
  put 1 into answ               (p. 415)
  repeat with i = 1 to exp      (p. 593)
    multiply answ by num        (p. 444)
  end repeat
  return answ                   (p. 561)
end power
```

The following commands show how the functions defined above might be called.

```
writeln(power(5,3));

put power (5,3) into field 1 (p. 561)
```

Similar to Pascal, HyperTalk provides for user defined subroutines, procedures or handlers with value parameters. HyperTalk also provides for both local and global variables. Local variables do not have to be declared in HyperTalk. Global variables must be declared in all subroutines or handlers which access them. In contrast with Pascal, HyperTalk handlers must return values through global variables since there is no provision for variable parameters. The following example procedure and handler compute the power of a number similar to the functions shown above.

```
procedure power (num, exp: integer; var answ:
    integer);
  var i:integer;
  begin
    answ:=1;
    for i:=1 to exp do
        answ:=answ*num;
  end;

on power num,exp
  global answ                   (p. 457)
  put 1 into answ               (p. 415)
  repeat with i = 1 to exp      (p. 593)
    multiply answ by num        (p. 444)
  end repeat
end power
```

The procedure or handler defined above would be called using the following commands. In the HyperTalk example, the global variable "answ" must be declared in any handler which calls the subroutine "power."

```
power (5,3,result);
writeln (result);

global answ
power 5,3
put answ into field "output"
```

Summary and Conclusions

Based on the comparisons presented above between Pascal and HyperTalk, it should be clear that HyperTalk is indeed a powerful language with many similarities to Pascal. It provides sophisticated sequence control structures, user defined subroutines and functions with parameter passing, a variety of data structures, powerful math operations and functions, string manipulation functions, text display, graphics and sound commands, and disk file input/output routines. Not shown in the comparison is HyperTalk's ability to call subroutines and functions recursively. These similarities should make it possible for Pascal programmers to become proficient in HyperTalk without undo effort.

In addition to the similarities with Pascal, HyperTalk also has some object oriented programming features similar to languages such as SmallTalk and some list processing features similar to languages such as Logo and Lisp. HyperTalk contains several other commands that have no Pascal equivalent, and therefore are not included in this comparison. Two of these which deserve special mention are the "edit script" and "do" commands. "Edit script" can be used to create and modify scripts from within a HyperTalk script, while the "do" command can be used to execute commands stored in a container such as a variable or field.

The HyperCard environment also provides a text editor and MacPaint type graphics tools. These tools, along with the capability to import text and graphics, makes it possible to easily create screen displays without having to rely on complicated programming commands.

HyperCard with HyperTalk does indeed provide a powerful and exciting programming environment. There are limitations such as screen size, lack of color, no commands for creating menus and dialog boxes, and the speed limitations of an interpreted language. However, many of these limitations may be rectified in future versions of HyperCard, or through the use of external commands written in other languages.

HyperChat™
XCMD Corner

Donald Koscheka
Apple Computers, Inc.
MacTutor Vol. 4 No. 9

Importing Text Into Hypercard

A new controversy seems to be emerging in the Hypercard community. Some Hypercard pundits are discouraging the use of XCMDs and XFCNs in stack design.. Their most convincing argument is that those of us who jump into writing XCMDs aren't giving ourselves an opportunity to see if HyperTalk can perform the task, perhaps equally as well as an XCMD.

I frequently consider writing an XCMD solution to a programming problem without first considering whether Hypertalk can do the same job for me. Recently, I needed to import Microsoft WORD files into Hypercard. What a wonderful opportunity to write an XCMD!

When I sat down to write the script to invoke the XCMD, I realized that I could write the entire WORD import routine in HyperTalk. Ed Wischmeyer of Apple Computer Inc. pointed out that although fields in HyperTalk prefer to see straight ASCII text, there is no such restriction on the contents of containers. Hypercard also allows you to open and read any file type you want; you aren't restricted to reading text files. Of course, you need to figure out how to translate what's in that container into a format that can be presented in a field.

The hard part of importing text from a Word file is not reading the data into hypercard but rather figuring how Word stores its text. By committing the import code to a simple Hypertalk script, I could concentrate my efforts at decoding Word's file format.

To simplify my search through the file format, I made the assumption that I could ignore any formatting information such as rulers, font and style changes. I was after was the text portion of the file only. This turns out to be a valid assumption since I wanted to import the file into a Hypercard field as text.

Finding the text was a snap with John Mitchell's "FEDIT+". I created a Word file using WORD and then examined it in FEDIT+. I noticed that the text always started at location 256 in the file. Since the size of the file was larger than the size of the text plus this 256 byte header, I needed to determine where the end of text occurred (assuming that the formatting and ruler information follows the text in the file). Since I knew how long the text was, I again used FEDIT+ to search the 256 header portion of the file. This time I was looking for any portion of the header that contained a count of the number of bytes in the text. Since I knew that my file contained exactly 100 characters (bytes), all I had to do was find this number somewhere in the header portion of the file. I found something close to what I was looking for at offset 16 in the file. This location corresponded to the number of characters in the text portion of the file plus 256 which was the length of the header.

The creators of Microsoft Word may be reading this and wondering why I'm assuming that the text size is a 16-bit entity rather than a 32 bit number. I'm not. Since Hypercard text fields are currently limited to 32K bytes, and since I knew none of my word files were longer than this, I'm only interested in the low-order word of the text length.

Reading the text portion of a Microsoft Word file into a hypercard container requires the following steps: (1) Position the mark at byte 16 of the file. Read the byte at this position and multiply it by 256 making it the high-order half of the file length. Read the next byte and add it to the hight-order half of the length. Move 238 more bytes into the file (16+2+238 = 256). This is the start of the text portion of the file. Read the number of bytes calculated minus 256. The IT container gets the imported text.

The Hypertalk script in listing 1 performs the above steps for importing up to 16K bytes of text from a Word file. I use Steve Maller's "FileName" XCMD to display WORD files only in the GetFile dialog and to get the full pathname of the file from the user. This script reads in the text without any looping so an XCMD may not speed things up enough to be warranted.

```
on mouseup
  put filename("WDBN")into filename
  if filename is not empty then
    open file filename - filename is the full pathname of a WORD
file
    read from file filename for 16 - move file mark to the text
length word
    read from file filename for 1  - read the upper half of the
length
    put chartonum( it ) * 256 into filesize - shift up by 8 bits
    read from file filename for 1  - get the lower half of the
length
    add chartonum( it ) mod 256 to filesize
   read from file filename for 238- move to start of text in the
file
    read from file filename for filesize-256 - read in the text
    close file filename - IT now contains the imported data.
  end if
end mouseup
```

Listing 1. Script to Import Text from a Microsoft Word File

Not all file formats can be imported quite so simply.

Macwrite uses a packed text format, storing one or two characters per byte using a simple compression scheme.

Because the text is compressed, we can't just read the file into a container and return the result to Hypercard. We must first decompress the file a byte at a time. Such a process suggests looping and loops, as we know, are not particularly fast in HyperTalk. Although the decompression can be performed in a hypertalk script, we can write an xcmd that performs the decompression faster.

The key to reading in a MacWrite file is understanding that Macwrite stores its data by paragraph. Whereas Word files are clearly divided between the text and formatting information, Macwrite stores formatting information for each paragraph at the end of the text for that paragraph. Hypercard doesn't do formatted text; we want to ignore the formatting information at the end of each paragraph. Our algorithm then becomes a loop that reads in a paragraph at a time, decompresses the text for that paragraph ignoring the formatting information. This process is repeated for each paragraph in the file.

One small "gotcha" to this approach stems from the fact that Rulers and pictures are also considered paragraphs. When we encounter either of these objects, we just move on to the next paragraph.

Listing 2 depicts the code for this XFCN. I chose "C" because pointer arithmetic is easier to perform in "C" and because last month's example was written in Pascal. I made every attempt to keep the "C" isomorphic to a Pascal program so that you can easily convert the code to Pascal.

Finding the paragraph information in the file requires a little arithmetic. Bytes 2-3 in the file tell us how many paragraphs the main document contains (MacWrite makes a distinction between the main document, the header document and the footer document. For our purposes, we only want to read in the main body of text) If bytes 2-3 contain a 5 then there are 5 paragraphs in the main document.

For each paragraph, MacWrite stores an information array. We start reading the information arrays at the file position pointed to in file offset $108. An information array is an array of 16-byte elements that tell us something about each paragraph. The first two bytes in the information array tell us whether the paragraph contains text, a ruler or a picture. If this value is positive the paragraph contains text, if this value is 0 or negative the paragraph is a ruler or a picture respectively and we can ignore it.

Offset 8 in the information array contains a status byte that provides some information about the text. If bit 3 is set, the text in this paragraph is compressed. Bytes 9-11 tell us the absolute file offset for the start of the data in the paragraph and bytes 12-13 contain the length of the data (paragraph addressing is 24 bits and each paragraph contains up to 64K of characters or data). The trick is to read in the number of characters indicated in the information array, determine if the paragraph contains text and, if so, decompress the text if it's compressed.

Once we read in the paragraph, we get some more information. The first two bytes of the paragraph tell us how many characters of text will appear in the decompressed paragraph. Following the text on an even word boundary is the formatting information for the paragraph which we ignore in this example.

MacWrite's text compression is based on a letter frequency scheme stored as STR resource #700 in MacWrite's resource fork. For English, this string contains " etnroaisdlhcfp". Macwrite maps these characters onto the array [$0..$F]. The space character ($20) has a value of 0, letter "e" has a value of 1, "t" a value of 2 and so on. Since any number less than $F can fit into a nibble, the word "eels" can be represented as "$11A8" rather than the byte-wide representation of $65656C73. In this example, we realize a 50% space saving (the best case for this algorithm).

This compression scheme only works for lower-case letters since 4 bits is not enough information to code for word frequency and case for the 14 most popular letters. This scheme also doesn't compress non-alphabetic characters such as numerals and punctuation marks. In these cases, the 16th array element, $F, is used as a flag to tell indicate that the next 2 nibbles represent one character. "Then" would be coded as $F55906. Note that the letter "T" crosses byte boundaries, the top nibble is in byte 0 and the lower nibble is in byte 1. This is of no consequence to the algorithm.

Armed with this information, you should have little trouble understanding the XFCN. In fact, I hope you find it useful and informative! (Next month: printing from XCMDs).

```
/*************************\
*     file:       MWRead.c    *
*                                 *
* an XFCN that imports text       *
* directly from a MacWrite file   *
* whose full pathname is passed   *
* as an input parameter.          *
*                                 *
* ________________        *
* To Build this file:             *
*                                 *
*     C -q2 -g MWRead.c           *
*                                 *
*     link -sn Main=MWRead ∂      *
*     -sn STDIO=MWRead ∂          *
*     -sn INTENV=MWRead ∂         *
*     -rt XFCN=3Ø1 ∂              *
*     -m MWREAD MWRead.c.o ∂      *
*     "(CLibraries)"CInterface.o ∂ *
*     -o "your stack name"        *
*                                 *
* ________________        *
* By: Donald Koscheka             *
```

```
* Date:     2-July-1988              *
* ©   1988, Donald Koscheka          *
*           All Rights Reserved      *
*                                    *
* ——————————————      *
\*************************/

#include    <Types.h>
#include    <OSUtils.h>
#include    <Memory.h>
#include    <Files.h>
#include    <Resources.h>
#include    "HyperXCmd.h"

#define     INFOPOS     0x00000108
#define PPOS            0x00000002
#define     COMP        0x0008


/* ——————————— */
/* Define the structure of an */
/* information array element  */
/*                            */
/* pHite is positive if this  */
/* info array points to text, */
/* ignored otherwise.         */
/*                            */
/* fPos is the absolute file  */
/* position of the start of   */
/* the paragraph in the file  */
/*                            */
/* fLen is the total length   of    */
/* the file including formats */
/* ——————————— */

typedef struct infoArr {
      short pHite;/* parag hite     */
      short pixels;/* ignore this   */
      long  pHand;/* ignore this    */
      char  status;/* chk comprsn   */
      char  hiMark;/* msw of mark   */
      short loMark;/* lsw of mark   */
      short fLen; /* parag. len     */
      short fmat; /* ignore this    */
}infoArr;

/* ——————————   */

short       ReadFile();
Handle            DeCompress();


pascal void MWRead( paramPtr )
      XCmdBlockPtr      paramPtr;
/*************************
* In:ParamPtr:
*           pointer to XCMD param
*           block. params[0] is the
*           name of the macwrite file
*           to open.
*
* Out:ParamPtr->returnValue
*           empty if data could not
*           be read, text portion
*           of a  Macwrite document.
*************************/
{
short
                  ref,  /* file reference */
                  err,  /* io error */
                  vRef, /* vol reference  */
                  pcnt, /* #  paragraphs  */
                  tSiz, /* text length    */
                  loop; /* loop counter   */
long
                  fSiz, /* data size      */
                  iSiz, /* out data size  */
                  iMark,/* iarr file pos  */
                  fPos; /* para. offset   */
Handle
                  ImportText,
                  decomp,/* decompressed*/
                  temp;
infoArr
                  info;
char
                  *fName,
                  vName[32];

ImportText = nil;

if( paramPtr->params[0] != nil ){
HLock( paramPtr->params[0] );

GetVol( vName, &vRef );
fName = *(paramPtr->params[0]);
err = FSOpen( fName, vRef, &ref );
HUnlock( paramPtr->params[0] );

if( err == noErr ){
      ImportText = NewHandle( 0 );

      /* get paragraph count        */
      fSiz = sizeof( short );
      err=ReadFile(ref,fSiz,&pcnt,(long)PPOS);

      /* get infoArray position           */
      fSiz = sizeof( long );
      err=ReadFile(ref,fSiz,&iMark,(long)INFOPOS);

      /* read in the paragraphs           */
      for( loop = 0; loop < pcnt; loop++){
            fSiz = sizeof( infoArr );
            err=ReadFile(ref,fSiz,&info,iMark);

            if( info.pHite > 0 ){
                  /* paragraph is text    */

                  /* calc text position   */
                  fPos=(info.hiMark<<0x10)+info.loMark;

                  /* get the char count   */
                  fSiz = sizeof( short );
                  err = ReadFile(ref,fSiz,&tSiz,fPos);

                  /* read in the text     */
                  temp = NewHandle((long)tSiz);
                  HLock( temp );
                  fPos += 2;
                  fSiz = (long)tSiz;
                  err = ReadFile(ref,fSiz,*temp,fPos);

                  if( info.status & COMP ){
                  /* paragraph is compressed    */
                        HLock( temp );

                        decomp = DeCompress( *temp, tSiz );

                        HUnlock( temp );
                        DisposHandle( temp );

                        temp = decomp;
                        tSiz = (short)GetHandleSize( decomp );
                  }/* if( info.status & COMP ) */

                  iSiz = GetHandleSize( ImportText );
                  fSiz = (long)tSiz;
                  SetHandleSize( ImportText, iSiz+fSiz );
```

```
                BlockMove(*temp,(*ImportText)+iSiz,fSiz);

                HUnlock( temp );
                DisposHandle( temp );
            }/* if( info.pHite > 0 ) */

            iMark = iMark + sizeof( infoArr );

        }/* FOR paragraph count */

        iSiz = GetHandleSize( ImportText );
        SetHandleSize( ImportText, iSiz+1 );
        *((*ImportText)+iSiz) = '\0';

        FSClose( ref );
        FlushVol( nil, vRef );
    }/* if file opened ok              */

    paramPtr->returnValue = ImportText;
    }

    }

    short       ReadFile(ioRef,siz,buf,from)
    short           ioRef;
    long            siz;
    char            *buf;
    long            from;
    /*************************
    * read cnt bytes from the file specified by parms and put
    * the data into the buffer pointed to by buf
    *
    * ioRef     = file reference number
    * siz = number of bytes to read
    * buf = where to read in to
    * from      = where in file to read from
    *
    * from is the file mark relative to the start of the file from
    * which the read is to start.
    *************************/
    (
        short       err;

        err = SetFPos( ioRef, fsFromStart, from );
        if( err == noErr )
            err = FSRead( ioRef, &siz, buf );
        return( err);
    }

    Handle DeCompress( inp, expcnt )
        char  *inp;
        short expcnt;
    /************************
    * Decompress the input handle's data (inH) and put the result
    * in the output Handle (outH).  outH is sized properly and
    * we use the following scheme:
    *
    *0 1 2 3 4 5 6 7 8 9 A B C D E F
    *_ e t n r o a i s d l h c f p !
    *
    * where     _ = SPACE
    *           ! = not compressed
    *
    * Consult MacWrite resource str #700 for the decompression
    * string in your file (different for other languages).
    * cycle through until the decompressed string count
    * matches the expected count
    ************************/
    (
        short           chcnt;
        register    char  *op;
        register    char  hiNib;
        register    char  loNib;
        char            dc[16];
        Handle          outH;

        outH = NewHandle( (long)expcnt );

        dc[0]  = 0x020;
        dc[1]  = 'e';
        dc[2]  = 't';
        dc[3]  = 'n';
        dc[4]  = 'r';
        dc[5]  = 'o';
        dc[6]  = 'a';
        dc[7]  = 'i';
        dc[8]  = 's';
        dc[9]  = 'd';
        dc[10] = 'l';
        dc[11] = 'h';
        dc[12] = 'c';
        dc[13] = 'f';
        dc[14] = 'p';

        HLock( outH );
        op = *outH;
        chcnt = 0;

        while( chcnt < expcnt ){
            hiNib = loNib = *inp++;
            hiNib = hiNib >> 0x04;
            hiNib &=  0x000F;
            loNib &=  0x000F;

            if( hiNib < 0x0F ){
                *op++ = dc[hiNib];
                chcnt++;
                if( loNib < 0x0F ){
                    *op++ = dc[loNib];
                    chcnt++;
                }
                else{ /* next BYTE is a char */
                    *op++ = *inp++;
                    chcnt++;
                }
            }
            else{
                /* next 2 nibbles represent   */
                /* a complete char which*/
                /* is on odd-nibble bounds    */

                *op   =     loNib << 0x04;
                hiNib = *inp++;
                loNib = hiNib & 0x000F;
                hiNib = hiNib >> 0x04;
                hiNib &= 0x0F;
                *op   = *op | hiNib;
                op++;
                chcnt++;

                if( loNib < 0x0F )
                    *op++ = dc[loNib];
                else
                    *op++ = *inp++;
                chcnt++;
            }
        }
        HUnlock( outH );
        return( outH );
    }

    #include <XCmdGlue.inc.c>
```

Listing 2. XFCN to Import trxt from a MacWrite Document

© The Definitive MacTutor, Vol. 4

HyperChat™

Happy Birthday

Fred Stauder
HyperChat Editor
Zurich, Switzerland
MacTutor Vol. 4 No. 10

on HyperChat

Happy Birthday HyperCard

MacWorld once again has brought us HyperCard–this time in it's first birthday shoes. There were forums dedicated to HyperCard and a first birthday extravaganza party which included a giant cake and a HyperCard time capsule.

The exciting thing at the show was a trend to expand the uses HyperCard.

Apple

Bill has done some more magic by creating Hyperscan, which is a stack that controls the new Apple scanner. Apple has also released through APDA a serial toolkit that allows you to access the serial ports through a series of XCMD's. Also released is a Laserdisc driver Interactive video toolkit that allows you to build videodisc stacks, and a HyperAppletalk driver that allows you to make networked HyperCard Stacks (this was written by Don Koscheka). Needless to say we will have some great articles on this in the future.

Versacad

A very novel way of using HyperCard was implemented by Versacad. They open Versacad files with HyperCard and get a list of objects in a drawing (HyperCard can open any file, and information can be written or retrieved by knowing the offset of items), then they use HyperCard to do a cost analysis of the project. For example, you put in the cost of each part of a motor, and HyperCard does the calculations.

Bright Star

Bright Star technology have released their HyperCard Face driver (see May issue) called HyperAnimator. This is the type of technology that will lead us to the Knowledge navigator. I will be showing you how to use this technology in innovative ways in a later issue. (See Figure 1).

ACM

I was at SIGGRAPH just before MacWorld, and it was really interesting to see the Mac being accepted warmly by the graphics crowd. The Association of Computer Machinery (ACM) has released a great Database of HyperText articles using HyperCard, a valuble resource. (See Figure 2).

Stack Starter

This is a stack which has the best collection of innovative buttons and controls that I have seen, including incredible animated icons with sound effects. This stack was written by Robertson Smith and is Shareware available from Cumputerware and BB's. (See Figure 3).

Amanda Stories

Amanda Goodenough has brought out another story Called "Your faithful Camel goes to the North Pole." Amanda is a story herself. She is a person that was unknown in the computer industry a year ago and now is very involved in the industry. She created "Inigo get's out", and the response from children was so great, it made us adults look at what she was doing. She was a speaker at MacWorld and is praised for her stack simplicity. I will demonstarate some of the concepts of design that Amanda is trying to convey in an article in a later issue. I think we should take an example from Amanda that Simplicity and content are the keys to good stack creation.

Great Book

Gary Bond's XCMD's for HyperCard is a must if you are witing XCMD's; it is a good beginners guide and a great reference book.

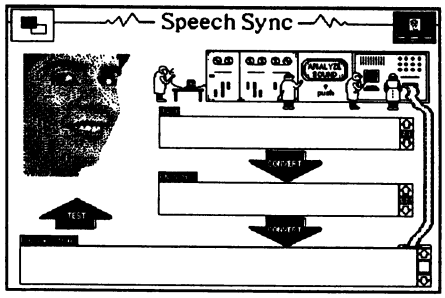


Figure 1. HyperAnimator From Bright Star

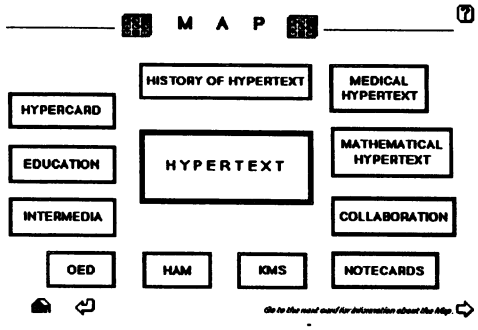


Figure 2. ACM

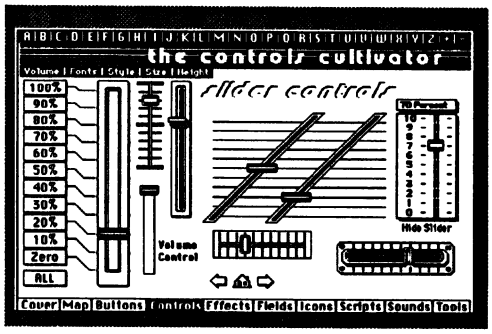


Figure 3. Sample Controls

HyperChat™

XCMD Cookbook: Print Manager

Donald Koscheka
Apple Computer
MacTutor Vol. 4 No. 10

Print Manager Access

Printing may not come to mind as an area where writing an XCMD might be useful since HyperCard prints card reports. Perhaps you don't like the reports that HyperCard prints out, or you want to set your own margins or draw borders around each field that's printed out. XCMDs provide access to the ToolBox Print Manager, empowering you with the capability of performing custom printing in HyperCard. The Print Manager, then, becomes a good candidate for XCMD exploration.

Reporter.P

This month's XCMD, Reporter.P (see Listing 1) is a simple text printer that you can adapt to your own needs or use as a template for further exploring Macintosh Printing. As always, we start with the XCMDBlockPtr as the interface to Hypercard. We pass a handle to the text in the first parameter. We could use the other parameters to pass formatting information. Parameter 2, for example, might be set to tell us to draw a border around the page. Parameter 3 might contain the margins in page coordinates.

The algorithm we use goes like this: Move to the top of the page and initialize a rectangle, lineRect, whose height is the height of the text to be printed and whose width is zero. Step through the text to be printed accumulating words (text separated by "white space" or punctuation). If the width of this word plus the width of linerect is less than the page width, add this line to the current line; otherwise, move down to the next line and add the current word to the start of the next line. If any part of the next line is below the bottom margin of the page, eject the page and reset lineRect's top to the top of the page.

Scoping

One of the nice features of writing XCMDs in Pascal is the scoping feature of procedures and functions. Pascal allows us to define procedures and functions within the main procedure, in this case, the procedure "Reporter". This is useful because it allows the callbacks and the nested procedures to share reporter's local variables. When we invoke a callback, we don't need to pass the parameterBlocPointer. The callbacks can "see" this pointer as if it was a global data declaration. Another nice feature of writing XCMDs in Pascal is that we don't have to be fussy about the entry point. The variable scoping simplifies the compiler's job of resolving the entry point. In "C", subroutines follow the main body of the XCMD so that the compiler can use the start of the main body as the entry point.

Step Through

Reporter first checks the parameter block pointer to determine whether enough parameters were passed. Params[1] should be a handle to some text to print. If the handle is NIL, we don't have anything to print; otherwise, we open the printer driver, allocate a printer record handle (prRecHandle) and fill the record with the print defaults. If you wanted to allow the user to change the page style (e.g. page orientation), this is where you would invoke the printer's style dialog. We'll skip this step for now because we want the XCMD to print the document with a minimum amount of intervention from the user.

Next, we put up the job dialog which is the standard printing dialog that comes up when you select print from the menu. If PRJobDialog returns true, the user clicked the "OK" button and wants us to print the document.

PROpenDoc associates our printing with a grafport. This technique effectively hides the gritty details of printing from the application. As long as we draw to the grafPort returned by PROpenDoc, the underlying code will convert our QuickDraw commands to Postscript for the LaserWriter and to a ribbon-eating bitmap for the Imagewriter.

Once the grafport is opened, we open the first page of the document with the PROpenPage and call PrintHandle to print the text. PrintHandle manages the opening, closing and ejecting of successive pages so when it returns all we need to do is close down the printing port with the PrCloseDoc call. Understanding this routine will take you a long way to understanding the how this printing business works.

Once we're done printing, we check to see if the document needs to be spooled out. Spooled files are saved to disk and printed by the call to PrPicFile. This statement insures compatibility with print spoolers that will be looking for picture files to print.

The actual printing is performed by the procedure PrintHandle. For the sake of argument, we set the font to nine point Geneva. A better approach would be to pass font information as a parameter to the XCMD. GetFontInfo tells us the height of the text since that will also be the height of a line on the page. We then calculate some rectangles and set the coordinates of our lineRect variable. Note that lineRect starts off at the top left of the page and has an initial width of 0. LineRect will expand to include the width of each word that we add to the line.

Locking down the text handle allows us to maintain pointers to the text. The first pointer, cpos, points to the current position in the text while wpos is used to point to the first character in the current word. A third pointer, tpos, is used as a "test pointer" and will remember the end of the current word.

The repeat loop in printhandle points tpos to the start of the next word (or the first character in the text if we're just starting the loop). If tpos is currently pointing to a line termination

© The Definitive MacTutor, Vol. 4

character (e.g. carriage return), we print the current line, move to the next line and reset wpos to point to the next word. Form feeds are handled in a similar fashion except that we eject the current page before continuing.

If tpos points to the nilChar (the ASCII character whose value is 0), we've printed all of the text in the input stream so we print the line, eject the page and exit the loop.

The otherwise clause checks to see if the next word in the input will fit on the current line. If so, add the word to the line and update the pointers accordingly. Since tpos points to the end of the current word, setting wpos equal to tpos causes us to "leap" over that word.

If the word doesn't fit on the line, we first print out the line and then move to the next line. After a word is "accumulated" into a line, we add the width of the word to the linerect.

The last routine of note, CalcNextWord, checks to see if the next character in the text is a word break character such as a space or punctuation. If so, we now have another word to add to the output line.

Summary

Reporter left justifies the text but you can easily add full justification. When you get to the end of the current line (i.e. wordsize + linerect > page width) calculate the number of pixels that would be needed to "fill" out the line. Subtract the current line's width (right-left) from the page width. If the current line is 2000 pixels wide and the page rectangle is 2500 pixels, then the number of pixels needed to fill out the line is 500. Divide this number into the number of spaces on the line and call SpaceExtra to space out the text.

You'll be surprised out how easy text formatting is with the Macintosh toolbox and I encourage you to use Reporter as a starting point for your explorations.

```
(*********************************)
(* File: Reporter.p              *)
(*                               *)
(* Prints the entire             *)
(* contents of the container     *)
(*                           *)
(* ------------          *)
(* In: params[1] = handle    *)
(* to the text to be printed     *)
(*                               *)
(* ------------          *)
(* © 1988, Donald Koscheka,      *)
(*All Rights Reserved            *)
(* ------------          *)
(*********************************)

(*****************************
    BUILD SEQUENCE

pascal Reporter.p
link -m ENTRYPOINT -rt ∂
XCMD=6555 -sn Main=Reporter ∂
Reporter.p.o ∂
"(Libraries)"Interface.o ∂
"(PLibraries)"Paslib.o ∂
-o "(xcmds)"testxcmds

*****************************)
```

```
{$S Reporter }

UNIT Donald_Koscheka;

{—INTERFACE—}

INTERFACE

USES
  MemTypes, QuickDraw, OSIntf,
  ToolIntf, PackIntf, HyperXCmd,
  PrintTraps;


PROCEDURE EntryPoint(pPtr:XCmdPtr);

{—IMPLEMENTATION—}

IMPLEMENTATION

{$R-}
CONST
  CARD   = TRUE;
  BKGND      = FALSE;
  NILCHAR    = $00;
  CR     = $0D;
  TAB    = $09;
  SPACE      = $20;
  FF     = $0C;
  LINEFEED   = $0A;
  QUOTE      = $22;
  COMMA      = $2C;
  PERIOD = $2E;
  PAREN      = $28;

TYPE
  Str31 = String[31];

PROCEDURE Reporter(pPtr:XCmdPtr);FORWARD;

{—EntryPoint—}

PROCEDURE EntryPoint(pPtr: XCmdPtr);
BEGIN
  Reporter(pPtr);
END;


{—Reporter—}

Function CalcNextWord(VAR wPtr:Ptr):INTEGER;
(**********************
* Given a pointer to a
* word, calculate the
* length of the next word
* in the run.  The length
* is determined by adding
* the width  of each character
* together until a word
* break is hit.
*
* The difference between
* tpos and wpos is always
* one word (including the
* sticky characters)
*
* IN: Pointer to text
* wPtr == pointer to NEXT word in
* run (or NIL if No next word)
*
* OUT: Width of the next
* word in the line
**********************)
VAR
```

```
  done : BOOLEAN;
  wLen : INTEGER;
BEGIN
  done := FALSE;
  wLen := 0;

WHILE NOT done DO
BEGIN CASE wPtr^ OF

CR,FF,LINEFEED,NILCHAR:
            done := TRUE;
TAB:
  BEGIN
  WLen:=wLen+CharWidth( chr(SPACE) );
  done := TRUE;
  END;

  SPACE, QUOTE, COMMA, PERIOD:
  BEGIN
  wlen:=wLen+CharWidth( chr(wPtr^));
  done := TRUE;
  wPtr:=Pointer( ORD(wPtr) +1);
  END;

OTHERWISE
  BEGIN
  WLen:=wLen+CharWidth( chr(wPtr^));
  wPtr := Pointer( ORD(wPtr) + 1 );
  END;
END; {*** CASE wPtr^ OF ***}
END; {*** WHILE NOT done ***}

IF wPtr^ = 0 THEN wPtr := NIL;

  CalcNextWord := wLen;
END;


Procedure PrintHandle( hand : Handle; printer : TPPrPort; Prec :
THPrint );
(**********************
* Print the data passed
* as a handle and
* using the given printer
* port.
*
* Prints the data into
* the current port and
* handles word wrap and
* page breaks.
*
**********************)
VAR
  done   : BOOLEAN;
  lineHite,
  wordSize    : INTEGER;
  pageHite,
  pageWidth   : INTEGER;
  num    : INTEGER;
  cPos   : Ptr;
  wPos   : Ptr;
  tPos   : Ptr;
  lineRect    : Rect;
  fontInfo    : FontInfo;

PROCEDURE EjectPage;
(****************************
* Eject the current page and
* adjust the rectangle
* accordingly
*
****************************)
BEGIN
  PrClosePage( printer );
  PrOpenPage( printer, NIL );

{*** Opening a new page yields ***}
{*** a new grafport, reset it  ***}
  TextFont( GENEVA );
  TextSize( 9 );
  GetFontInfo( FontInfo );
  lineHite := fontInfo.ascent +
      fontInfo.descent +
      fontInfo.leading;

WITH Prec^^.prInfo.rpage DO
BEGIN
lineRect.Top := top;
lineRect.Bottom:=lineRect.top
+lineHite;
lineRect.Right := lineRect.left;
Moveto(lineRect.left,
lineRect.bottom );
END;
tpos := Pointer( ORD( tpos ) +  1 );
END;


PROCEDURE NewLine;
(*******************************
* Move to a new position on the
* page.
*******************************)
BEGIN
  WITH lineRect DO
    BEGIN
      top    := top + lineHite;
      bottom := bottom + lineHite;
      MoveTo( left, bottom );

      IF bottom > pageHite THEN
        EjectPage;

  Right := Prec^^.prInfo.rpage.left;
    END;
END;

PROCEDURE DrawLine;
(*******************************
* Draw the current line please
*******************************)
BEGIN
  DrawText( cPos, 0, num );
  lineRect.Right := lineRect.left + wordSize;
  tpos := Pointer( ORD( tpos ) + 1 ); { debug }
  num := INTEGER( ORD(tPos) - ORD(wPos) );
  cPos := wPos;
  wPos := tPos;
END;


{— Reporter —}
BEGIN
  TextFont( GENEVA );
  TextSize( 9 );
  GetFontInfo( FontInfo );
  lineHite := fontInfo.ascent +
        fontInfo.descent +
        fontInfo.leading;

{*** get information about page ***}
  WITH Prec^^.prInfo.rpage DO
    BEGIN
      pageHite        := bottom - top;
      pageWidth  := right-left;
      lineRect.top    := top;
      lineRect.bottom:= top +
    lineHite;
```


© The Definitive MacTutor, Vol. 4

```
      lineRect.left   := left;
      lineRect.right  := left;
      MoveTo( lineRect.left,
lineRect.bottom );
    END;

  Hlock( hand );
  cPos := hand^;
  wPos := cPos;
  num  := 0;
  done := false;

  REPEAT
    tPos := wPos;

    CASE tPos^ OF
      LINEFEED, CR:
        BEGIN
            DrawLine;
            NewLine;
            wpos := tpos;
        END;

      FF:
        BEGIN
            EjectPage;
            wpos := tpos;
            lineRect.right := lineRect.right + wordSize;
        END;

      NILCHAR:
        BEGIN
            DrawLine;
            done := TRUE;
        END;

      OTHERWISE
        BEGIN
            wordSize := CalcNextWord( tPos );
            IF( wordSize + lineRect.right ) < pageWidth THEN
                BEGIN
                    num := num + INTEGER( ORD(tPos) - ORD(wPos)
);
                    wPos := tPos;
                END
            ELSE
                BEGIN
                    DrawLine;
                    NewLine;
                END;
  lineRect.right:=lineRect.right + wordSize;
        END;

    END (** CASE ***);
  UNTIL done;

  PRClosePage( printer );
  HUnlock( hand );
END;
```

```
PROCEDURE Reporter(pPtr: XCmdPtr);
(**********************
* Print the data that's
* passed in as a
* parameter.
**********************)
VAR
  fieldPtr        : Ptr;
  fieldName       : Str255;
  fieldType       : BOOLEAN;
  fieldData       : Handle;
  prRecHandle: THPrint;
  prPort      : TPPrPort;
  myStRec         : TPrStatus;

{$I XCmdGlue.inc }

BEGIN
WITH pPtr^ DO
IF (paramCount <> 0) AND
(params[1] <> NIL) THEN
BEGIN

PROpen;
prRecHandle := THPrint(NewHandle(
  SIZEOF(TPRINT) ));
PrintDefault( prRecHandle );

IF PRJobDialog( prRecHandle ) THEN BEGIN
prPort := PROpenDoc( prRecHandle, NIL, NIL );
PrOpenPage( prPort, NIL );

PrintHandle( params[1], prPort, prRecHandle );

PrCloseDoc( prPort );

        IF(prRecHandle^^.prJob.bJDocLoop  =  bSpoolLoop)  AND
(PrError = noErr) THEN
  PRPicFile( prRecHandle, NIL, NIL,                N I L ,
myStRec );

END; (*** IF PRJobDialog ***)

  DisposHandle(Handle(prRecHandle) );
  PrClose;

END; (*** paramCount <> 0 ***)
pPtr^.returnValue := NIL;
END; (*** PROCEDURE Reporter ***)

END.
```

Listing 1. Reporter.P - A text printing XCMD.

end HyperChat

HyperChat™

XCMD Cookbook: Calling Conventions

Donald Koscheka
Apple Computer, Inc.
MacTutor Vol. 4 No. 11

```
on HyperChat
```

A Nice Chat

While doing the Sunday crossword puzzle recently, I was hit with a very interesting thought. One of the clues in this particular puzzle called for a synonym for "small talk". The answer was "chat". This struck me as odd because in many ways HyperTalk is very much the preferred object oriented programming language for the Macintosh. One of the earliest object oriented languages, as you know, is called "SmallTalk". By calling this column "HyperChat", Fred makes a rather obscure reference to HyperTalk's philosophical roots. Chat implies a looseness of speech, a vernacular that is easy to master. That is exactly what HyperTalk is – an easy to grasp interface to the Macintosh Toolbox. What HyperTalk lacks in power, it certainly makes up for in ease of understanding!

At the opposite end of the spectrum is Assembly language programming. This much maligned language conjures all sorts of anxieties in the minds of otherwise fearless programmers. Yet most professional programmers will admit that an understanding of assembly language can improve one's ability to write efficient code in a higher-level language as well as better understand those mysterious looking dumps that one gets when suddenly presented with a memory dump from TMON or MACSBUG.

Worse yet, those of us who choose to work in higher level languages provide a great disservice to the assembly language programmer by not showing them how to interface with our languages. HyperTalk documentation is certainly sparse in describing how an assembly language programmer can write an XCMD in assembly.

Assembly Interface

I started writing this month's column hoping to be of service to the assembly language programming community. I wasn't very far along when I realized that this column offers a second service, at no extra charge to the reader.

The process of showing you how to interface to HyperCard in assembly language also introduces the Pascal and C programmer to debugging in Macsbug or TMON (Forth programmers take heart: Jörg Langkowski's article in the December, 1987 issue of Mactutor provides you with the information you'll need to write XCMDs; even if you don't program in Forth-like languages, you should read Jörg's column; his insights into the Macintosh are often astonishing).

The most important reason to code in Assembly language is that it provides a rich opportunity to improve on the more generic code generated by Pascal and C compilers. Consider the following string comparison in Pascal:

```
len : INTEGER;
Match    : Boolean;
Str1: Str255;
Str2: Str255;

Str1 := 'Hello World';
Str2 := 'GoodBye Cruel World';


IF Length(Str1) <> Length( Str2 ) THEN
    match := FALSE
ELSE
BEGIN
    Match := True;  { assume they will match}
    k := 1;

    While (k <= Length( Str1)) AND (match) DO
    IF Str1[k] <> Str2[k] THEN
        match := False
    ELSE
        k := k + 1;
END;
```

Listing 1. String Comparisons in Pascal

String compares can be made more efficient than this in Pascal. I chose this example for its simplicity. In comparing strings, we must consider the end points, if the strings are not the same length, they are not equal. If you haven't been programming in Assembly language, you may not see how this routine could be made more efficient. One measure of a program's efficiency is a count of the number of bytes of instructions needed to execute the program. If you compile listing 1 and dump the object code, you would discover that the while loop requires 82 bytes of instructions to execute. Assembly language programmers know by instinct that 82 bytes is too much code to perform a single string compare.

An assembly language equivalent of the while loop in listing 1 can be shrunk by an order of magnitude as in listing 2. If you're trying to speed up a particularly slow loop, a few bytes of well-written assembler might be just what the doctor ordered. But optimizing code is just one reason to familiarize yourself with assembly language. A more compelling reason for the high-level language is that an understanding of assembler will help you make sense of your TMON or MacsBug dumps.

```
                                ; A0, A1 point to the 2 strings
    Move.b    (A0)+, D1         ; get the length of string 1
    Move.b    (A1)+, D2         ; get the length of string 2
    Cmp.b     D1,D2             ; are they the same length?
    Beq       CompareChars      ; yes, go ahead and compare them
    Move      #0, D0            ; set the result to false
    Bra       Done              ; and exit

CompareChars                    ; compare Str1<->Str2
    Cmp.b     (A0)+,(A1)+       ; do the characters match?
    Beq       Done              ; yes, see if we're at end of
```


© The Definitive MacTutor, Vol. 4

```
string
   Dbra     D1, CompareChars
   Move     #1, D0        ; they match, set the result true

DONE        sne D0        ; Result is true if strings match,
                false otherwise
```

Listing 2. String Compare in Assembler

Entering XCMDs

The real trick to writing an XCMD in assembly language is understanding that HyperCard expects to see an XCMD that was generated by the Pascal compiler. This implies that parameters are pushed on the stack from left to right and that subroutines are responsible for removing the pushed parameters from the stack. XCMDs receive only one parameter so the push order isn't important. What is important is remembering to remove that parameter from the stack before returning to HyperCard. Listing 3 is the assembly language interface for XCMDs. The fields in the paramBlock record are identical to the Paramblock record in Pascal or C.

On entry to a subroutine in assembly language, the last item on the stack is the return address of the calling routine. Normally, when we are done with the subroutine, an rts (return from subroutine) instruction will pop this return address off the stack and into the Program Counter resuming execution at the instruction pointed to by that location. This won't do for Pascal routines since convention dictates that we also remove the parameters from the stack. One way to do this would be to pop the return address into a temporary register, say D0, remove the parameters from the stack by adding the size of the parameters to the stack pointer and then pushing the contents of D0 onto the stack and executing an RTS. A more efficient method exists: Pop the return address into an address register, say A0. Unbias the stack parameters by adding 4 to the stack (the size in bytes of the parameter block pointer). Finally, since register A0 contains the return address, execute the Jmp Indirect instruction on A0: Jmp (A0).

Globals in XCMDs

If your XCMD requires local variables, you're going to need a place to store them. Assembly language XCMDs bear the same restriction placed on high-level languages: you don't have access to the globals. A simple solution would be to allocate a handle large enough to store all your globals and keep that handle available in an address register. This works fine if the data is very static but not very well otherwise since you could easily run out of registers while manipulating even a small number of handles. Pascal and C use a more efficient approach, one that you've already seen if you've done any debugging in TMON or Macsbug.

On entry to the subroutine, we know that the stack pointer, register A7, already points to the next available space on the stack (the bottom of the stack). Why don't we allocate our data on the stack by pointing an address register, say A6, to the bottom of the stack, subtracting the number of bytes that we need for our locals from A7 effectively growing the stack by the amount we need (the stack grows downward). Now A6 points to our local

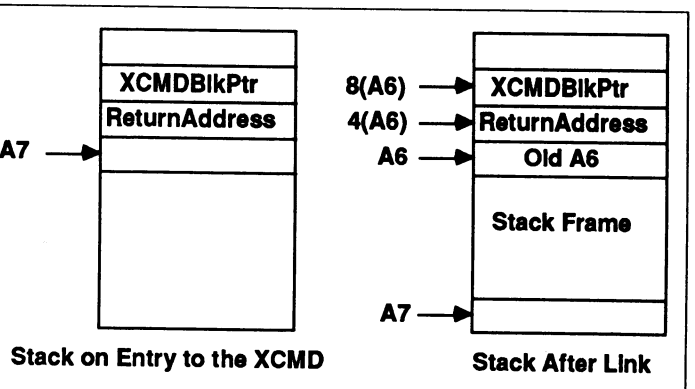


Figure 1. Link allows you to allocate storage on the stack.

variables and A7 continues its role as stack pointer below our local stack frame (figure 1).

The process of creating a stack frame can be performed with one assembly language instruction: link. Used judiciously, the link instruction buys us a whole lot more: it saves the old value of the address register and gives a reference point to the parameters passed by the caller.

By executing link A6,#LocalSize, before doing anything else, we set up up a stack frame at the stack bottom. If stacksize were set to zero, we wouldn't actually allocate any stack space for globals, but the instruction would still provide a payback. Because nothing else was put on the stack between the call to this routine and our link instruction, A6 also doubles as a pointer to our parameters! First, 0(A6) contains the previous value of A6 (can you think of anything this might be useful for?), next 4(A6) is the return address, and 8(A6) is the paramblock pointer passed to us by HyperCard. By putting 8(A6) into A3, we save the pointer to our parameters in an address register.

The offsets defined in the parameter block equates now become offsets off A3 for each field. The first field in the record is paramCount(A3) the count of the number of parameters in the params array. Accessing the handles in the params array poses another question. The first handle in the array is at params(A3). Getting to the other handles in the params array requires a straightforward application of arithmetic. By definition, handles occupy 4 bytes in the Mac. Any array element, i, is at offset (i-1)*4 byte in the array. We subtract 1 from the element number because array indices count from 0 not from 1. If we put the array index into D0, subtract 1 and multiply by 4 we have the offset from the beginning of the array. All we need to do is add this number to params[A3] and we have the handle. The 68000 has just an instruction: indexed addressing with offset. Here is how this instruction can be used to get the third parameter in the list:

```
Moveq     #3, D0   ; Access the third parameter in the list
Sub.w     #1, D0             ; arrays count from 0, not 1
Asl.w     #2, D0   ; shift left by 2 is same as multiply by 4!
Move.l    params(A3,D0.w), A0  ; A0 now holds the third handle
```

Since stack frames are oriented from the highest memory location they occupy to the lowest, local variables are always referred to by negative offsets. In Pascal, the following declaration

```
VAR
myInt    : INTEGER;
myLong   : LongINt;
myRect   : Rect;
```

would have the following counterpart in assembly language:

```
myInt       EQU -4        ; locals start at offset -4 in the
frame
myLong      EQU myInt-2   ; Integers are 2 bytes
myRect      EQU myLong-4  ; longs are 4 bytes
LocalSize   EQU myRect-8  ; rectangles are 8 bytes.
```

LocalSize is equal to 14 bytes. The stack frame is set up to read:

```
LINK    A6,#LocalSize     ; create the local variable pool.
Move.w  (A0),D0
```

If you're having a bit of difficulty with this material, try writing a simple XCMD in Pascal or C and disassembling it in TMON or Macsbug. You can do this by invoking the Debugger call as the first statement in you XCMD.

Exiting the XCMD

Leaving the XCMD requires a little house cleaning. First, we restore the registers that were saved onto the stack. Next, we execute an unlink (UNLK) instruction to undo the last Link instruction. At this point, the stack looks just like it did when we entered the XCMD. More importantly (A7) is the return address of the calling routine. Popping this value into A0 allows us to save the return address in a safe place so that we can remove the parameters from the stack. We know how much space the parameters take. The stack contains only one parameter, XCmdBlkPtr, whose length is four bytes. Adding 4 to A7 shrinks the stack to the right size. Now all that's left is to Jmp to the location pointed to in A0 and we're done!

Conclusion

If you're already programming in assembly language, this article is enough to get you started with XCMDs, especially if you're not familiar with Pascal calling conventions. If you're a Pascal or C programmer, the above discussion should help you to debug your code by explaining some of the assembly code you may have been looking at in TMON or Macsbug. In any case, understanding how compilers take your statements and convert them into machine executable code is a great way to learn more about the inner-workings of your programs and those seemingly mysterious bugs that just are not obvious in your Pascal or C source code. Hopefully, I've been able to fill some gaps in your debugging skills with this information. If the information in this article was useful, please let me know, I'll be happy to offer more information on assembly language and debugging techniques.

```
;******************************
;* File: SimpleXCMD.a         *
;*                            *
;* A simple XCMD written in   *
;* Assembly language to show  *
;* how XCMDs are written in   *
;* assembler...               *
;* ---------- *
;* By:   Donald Koscheka      *
;* Date: 16 July, 1988        *
;******************************

;******************************
;*              Build Sequence
;*
;* asm -w SimpleXCMD.a
;* link  -rt XCMD=1200 -sn Main=SimpleXCMD∂
;* SimpleXCMD.a.o∂
;* -o "YourStackName"
;******************************

;*** ParamBlock structure  ***
paramCount   EQU     0
params       EQU     paramCount+2
returnValue  EQU     params+( 16 * 4 )
passFlag     EQU     returnValue + 4
entryPoint   EQU     passFlag + 2
request      EQU     entryPoint + 4
result       EQU     request + 2
inArgs       EQU     result + 2
outArgs      EQU     inArgs + ( 8 * 4 )
pBlkSize     EQU     outArgs + (4 * 4 )

;*** ---------    ***
;*** THE LOCAL VARIABLES  ***
;*** (WILL GO ON STACK)       ***
;*** Note that the stack frame ***
;*** counts backwards from 0   ***
;*** so that the value of      ***
;*** LocalSiz will always be   ***
;*** negative                  ***

LOCALS       EQU     0
LASTLOCAL    EQU     LOCALS
LOCALSIZ     EQU     LASTLOCAL
;*** ---------    ***


SimpleXCMD   MAIN    EXPORT
;******************************
;* In:
;*
;*  0(A7) == Return Address
;*  4(A7) == ParamBlockPtr
;*
;* Link A6 to create a stackframe
;* that points to these vars.
;******************************

                ;*** Set Up Stack Frame
    LINK    A6,#LOCALSIZ     ; Size of the local frame
    MOVEM.L     D5-D7/A3/A4,-(SP) ; save some registers

                ;*** Get Pointer to paramblock
    MOVE.L      8(A6), A3        ; Point to parameters
    CLR.L       returnValue(A3)  ; set to "empty"
    TST.W       paramCount(A3)   ; Any Parameters?
    BEQ         DONE             ; no, just return

    ;*** Insert your code here.  If your XCMD doesn't take any
    ;*** parameters eliminate the atest on paramcount ...

DONE        ;*** Prepare for Return to HyperCard
    MOVEM.L     (SP)+,D5-D7/A3/A4 ; restore registers
    UNLK        A6               ; wipe out stack frame
    MOVE.L      (A7)+, A0        ; get the return address
    ADD.L       #4, A7           ; unbias the stack
    JMP         (A0)             ; return to HyperCard
    END
```

Listing 3. SimpleXCMD in Assembly Language

```
end HyperChat
```

HyperChät™

The Vertical Market

Fred Stauder
HyperChat Editor
Zurich, Switzerland
MacTutor Vol. 4 No. 12

on HyperChat
— HyperEditorial
HyperCard: what will the future hold?

HyperCard is just over a year old; some people say that it's not the panacea that everyone thought it was; others say it's the best thing since sliced bread.

What is it being used for? We don't see hordes of stackware, in the shelves at the local computer store, yet do we? Well it seems to me that Apple was right with it's obscure placement of HyperCard as System Software. Does HyperCard really fall into this category? HyperCard is used as controller for sound digitizers, MIDI devices, CD-ROM's, Laserdiscs, and Scanners. It also allows other programs such as Macromind's Videoworks to run as drivers. It can act as a finder replacement and much more.

With the introduction of MacPaint, the rest of the world started to play catch-up to develop paint programs. We shall see programs that have similarities to HyperCard emerging. Once again Apple was not that interested in the sales of MacPaint, it used it to "Change the World one Computer at a time" and drive it's market forward. It is interesting that both programs are by the same author, namely Bill Atkinson. HyperCard is not the "be all and end all"; it is merely the start of the next generation. We should call it the "HyperGeneration" or "MultiMedia Generation", just as MacPaint was the forerunner of the "Graphic Interface Generation"

College curricula are already offering courses in Hypertext and MultiMedia. We don't hear about the most interesting HyperCard applications because they are vertical. A few examples:

Techniques in surgery of the hand (utilizing scanned images and videoworks)
The electric cadaver(using laserdisc)
Genetic maps and molecular biology
Floorplans of large automotive factories
Artificial archeology.

This list shows the diverse nature of HyperCard. It is possible now to fill a lot of previously hard to access niches. Well what kind of vertical market opportunities are out there? Just look around; A catalog of parts and part numbers for regional offices, Training disks for employees. How do you get into developing vertical market applications? We have seen that the market defines that the content of a stack in particular (naturally a good interface is vital) is important, and, if you can produce a stack that is "Educationally Entertaining", all the better. For example, you are a HyperCard developer and a scuba diver; you might get together with an instructor to make a stack to teach diving. You can do it simply(text and some graphics), or animated graphics or videoworks animation(color and sound); if it gets too big, put it on a CD-ROM. You can even go as far as using video and putting it onto laserdisc. The extent and depth you wish to go to depends on the usual factors, time and money. The beauty of HyperCard is that if you only do a simple stack it can easily be expanded by someone else. There are literally thousands of opportunities waiting in the vertical market arena. I was recently in the Serengeti Wildlife Reserve in Tanzania Africa, on a Photographic Safari and I met a scientist that studied leopards. He told me he had all this video footage of the leopards; how they kill their prey and drag them into the tree tops to eat them, plus stills of bone fragments, and tooth pressure data obtained from the bones etc. He was saying wouldn't it be wonderful if you could put all this data from different media together and present it. When I told him about HyperCard and what it can do he nearly fell over backwards and told me I was a godsend. So you see it is not hard to find vertical markets even when you are on vacation.

A really good book that gives great insight into planning a HyperCard project is *Applied HyperCard: Developing and Marketing Superior Stackware* by Jerry Daniels and Mary Jane Mara it is published by Brady Software and comes complete with a disc.

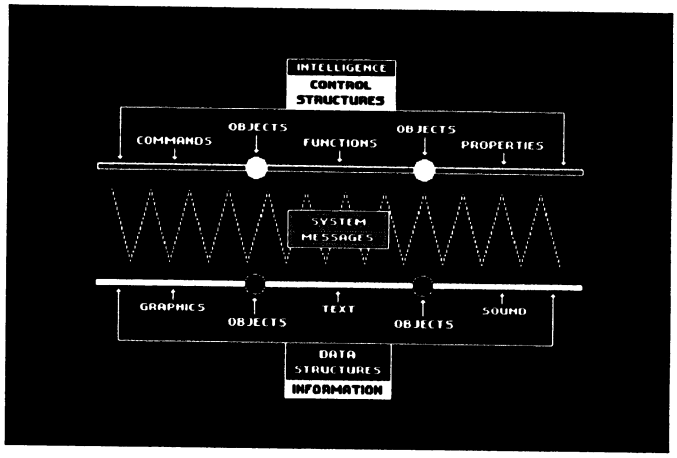


figure 1.

You may remember the authors made a stack called MacTV, a type of future look at what TV will be like when controlled by a computer. They take you from basic concepts of HyperCard and hypermedia through project management to marketing and distribution. The appendices list distributors, publishers, press contacts, user groups as well as a hypertalk glossary. This is a book that is ideal for the developer who is thinking of going commercial.

© The Definitive MacTutor, Vol. 4

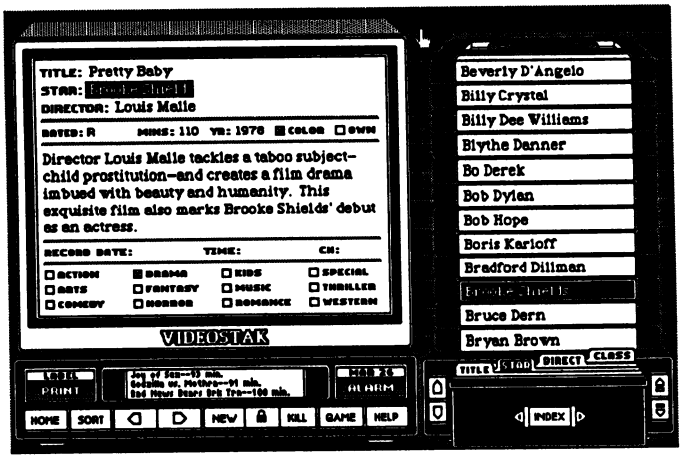


figure 2.

As mentioned earlier Apple has released a development package for Videodisc. It comes complete with drivers for most players, or you can write your own.

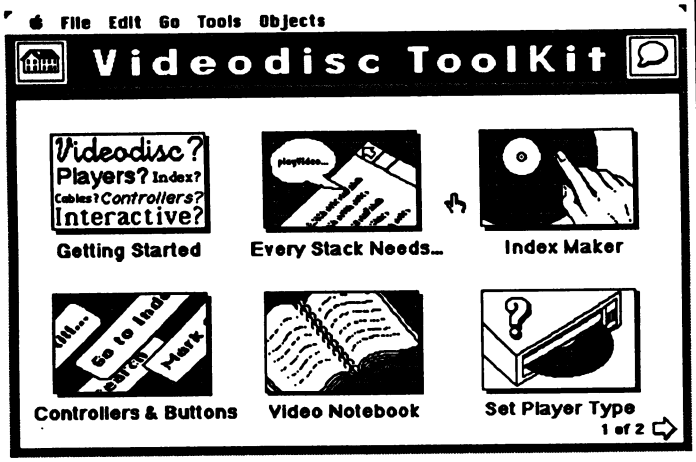


figure 3.

It also gives some great ideas for interface design with controllers. Such as the simple controller that looks like the control panel on a Videodisc player.

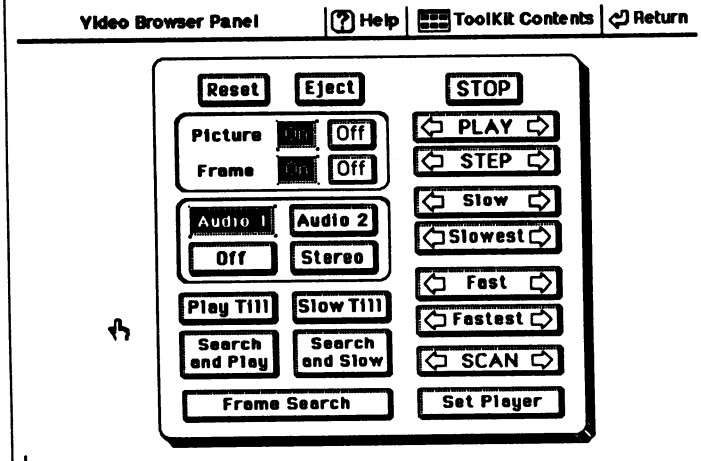


figure 4.

The human interface group at Apple did a great job with the gesture controller, when you are watching the video you can go back and forward using the mouse without taking your eyes off the screen. This package is well worth getting (through APDA). The investment in Videodisc technology is not that expensive at well under $1000 for player and monitor.

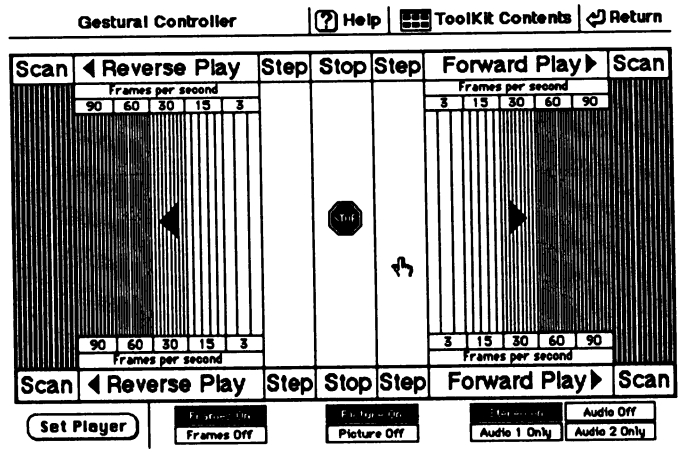


figure 5.

—HyperBugs

Dear HyperEd,

I have found an interesting bug (or maybe it is just a feature?) in HyperCard (versions 1.2 as well as 1.2.1, system version 4.3, finder 6.0 under or without multifinder 1.0).

From any stack, open the message box, then

1. Hide card window (from then, HyperCard loses control of your keyboard, even in blind-typing mode).

2. Hide the menubar and finally;

3. close the message box; from that point, the Mac is hung and I could not find any means to get control back other than rebooting. Even under multifinder, and using macsbug is of little help.

Sent in by Jean-Pascal J. Lange
Commission of the European Communities
Joint Research Center

—ScriptTips

This month I have written a script that will show all the hidden buttons and fields. I have used a single button and depending where you click it, it will do that operation. I will leave it as an exercise for you to try to improve the script so that it remembers which ones were originally hidden. This should be easy for the people that followed the AutoHyperEdit evolution.

```
—— ooooShowHiddens  by Fred Stauder  Oct 19880000 ——
- Hold the mouse down on cdBtn to show hidden card buttons
- it will show all of them or until you release the mouse
button

on mouseDown
  get trunc(((item 1 of the  clickloc) - (item 1 of the rect
of me)) /¬
  (((item 3 of the rect of me) - (item 1 of the rect of me)) /
```

```
4))

  if it is 0 then

    repeat with i=1 to the number of card fields
      if the mouse is up then exit repeat
      show card field i
    end repeat

  else if it is 1 then

    repeat with i=1 to the number of background fields
      if the mouse is up then exit repeat
      show background field i

    end repeat

  else if it is 2 then

    repeat with i=1 to the number of card buttons
      if the mouse is up then exit repeat
      show card button i
    end repeat

  else if it is 3 then

    repeat with i=1 to the number of background buttons
      if the mouse is up then exit repeat
      show background button i

    end repeat

  end if
end mouseDown

Non-Comercial use only - All Rights Reserved
Fred Stauder, HyperCard Editor, (HyperChat) MacTutor
—— ooooShowHiddens  by Fred Stauder  Oct 19880000 ——
```

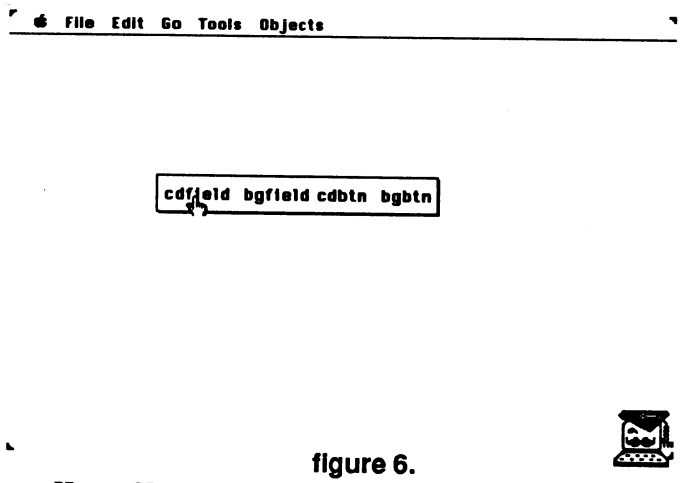
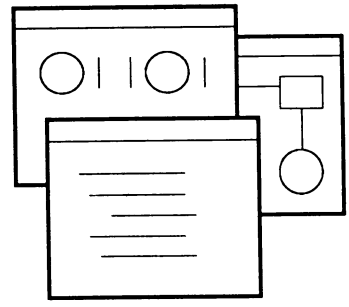
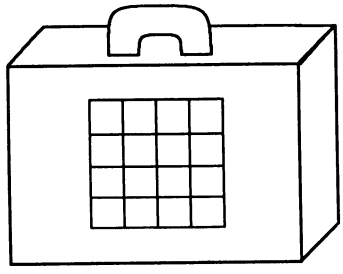
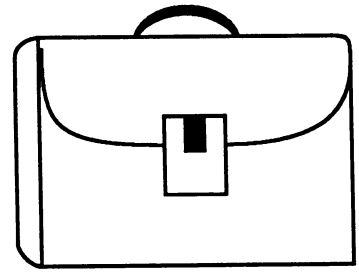


figure 6.
Have a HyperChristmass from the HyperChat Team!







More Articles

© The Definitive MacTutor, Vol. 4

MPW Workshop
Using Consulair C & TML in MPW

William Powell
Salt Lake City, UT
MacTutor Vol. 4 No. 1

William Powell is a candidate for a doctorate degree in Geophysics from the University of Utah, and is responsible for the design, documentation and delivery of software used to operate the Voyager spacecraft while a technical assistant at the Jet Propulsion Laboratory in Pasadena. He is presently doing research in the thermal physics of the Colorado Plateau

I am highly resistant to changing development systems, but after after all the fuss over MPW and reading J. West's MacTutor article [vol. 3, no. 2] on the resource tools in the Macintosh Programmer's Workshop (MPW), I decided that I couldn't live without this product. Along with everyone else, I have frequently experienced frustration from the limitations of earlier resource-building programs. I have also wished for a programmable shell on my Mac for certain data processing operations which I used to perform very conveniently on Unix systems. So I broke down and purchased the APDA release of MPW, and a hard disk to hold it. I have subsequently made some efforts toward integrating the MPW shell with older applications. I hope that some of my discoveries and command scripts may be generally useful to others who are using applications with MPW.

The programmable shell and tools of Apple's Programmer's Workshop (MPW) give a new dimension of flexibility to Macintosh development. The MPW brings the Macintosh a long step closer to the capabilities of more established development systems such as those available in the Bourne shell or C shell of Unix systems. This necessarily means that MPW has made a long step *away* from earlier Macintosh application–type development systems. MPW introduces a new class of compiled program, the MPW Tool, which interacts with its shell more intimately than earlier applications interact with the Finder and similar shells.

Bridging the gulf between the MPW shell and earlier Finder application development systems is the topic for this article. I will present examples for my development systems of choice, Consulair MacC and TML Pascal. There are three core issues in joining the MPW shell and application-type development systems. First is the problem of conveniently running applications in concert with programmable shell scripts and the automatic make utility. Second is the problem of writing a new class of program, called a MPW tool, using a development system which does not contain provision for such a thing. Third is object file compatibility with the MPW standard. This article will concentrate on the first item.

Many of the techniques discussed here are relevant to almost any programming of the MPW shell and will be of interest even if the reader does not need the sample command scripts for running development applications. This article includes examples of advanced MPW editor commands, examples of complicated shell quoting problems, and examples showing the use of shell variables. Information here about launching applications from the shell and about the distinction between windows and files is not found in the MPW Reference document. I also use the Make tool three times in this article — only one time involving building a program! I will assume that the reader is slightly familiar with the shell and has access to the MPW Reference document for more detailed background.

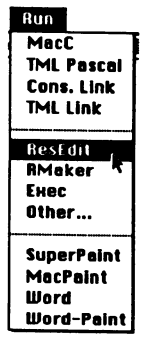


Figure 1. User-defined MPW menu for running applications.

Motivation

Why did I make the effort to mix older development systems into MPW when an assembler and C and Pascal compilers are already being distributed? One consideration is budget; it is hard to justify the cost of *another* C or Pascal compiler when the ones I have actually work just fine. Source portability is never perfect across Macintosh compilers and I haven't felt too anxious about converting yet another program to yet another compiler. Also, as noted by F. Alviani [MacTutor vol.3, no.4] the MPW system contains its own quirks, such as segmentation established in compilation rather than in linking.

There may be hardware considerations as well. The MPW documentation suggests that the MPW C compiler needs every byte of a 1M machine. This should be a serious consideration for those of us who develop in C on the pre-1987 Macs. It is good that the MPW system is forward looking in terms of newer machines, but many of us can't (or won't) upgrade our hardware and software as frequently as it is offered to the market.

On the other hand, the MPW shell offers the most flexible programmable shell and scripting abilities available for the Mac. The text editing features of the MPW shell are quite convenient, and more significantly, programmable. The first complete and extendable set of resource tools resides in the MPW. The programmable shell allows you to do things "your way" instead of locking you into someone else's concept of good development tools. It also provides a suitable environment for many less human–interactive data processing needs.

© The Definitive MacTutor, Vol. 4

So there are excellent reasons to add the MPW shell to the set of tools available to most Mac programmers and there are good reasons for some of us to keep our older development systems. The puzzle now is to find convenient ways to use the power and flexibility of the MPW shell in conjunction with older Finder-based development applications which have no facility to interact with the shell.

Applications and the MPW shell

The MPW shell is primarily a command interpreter, which parses lines typed by the user and interprets them as some sort of instructions to be performed by the computer. A simple command normally appears as a line of text. The first word on a command line is the name of the command to be performed — conventionally a verb describing the action performed (e.g. Count, Make, Delete, Open, etc.). Subsequent words on the command line are arguments which either indicate objects of the action (e.g. files to be read or written) or specify modes or options for the action (choices which would be selected with menus or modal dialogs in Macintosh applications). As with most programming languages, syntax is available to produce compound command statements from several simple commands as described here. The shell also allows the setting and testing of values of variables. When I refer to variables in the following discussion, I will use the notation {variableName}.

Commands are implemented in four different ways. Some are built into the code of the MPW shell application itself. Some are separately compiled programs called "MPW Tools" which run from within the shell, much as desk accessories run from within other applications. The name of a 'TEXT' file containing one or more lines which the shell can interpret as commands may itself be used as a command. Finally, for compatibility, the authors of the MPW shell allowed the names of Finder applications to be used as commands which cause the applications to start.

Applications and the Finder environment

Normally when an application runs, it starts off with a nearly clean slate. It initializes most of the ROM toolbox managers. The only high-level data in the application areas of memory which are normally available to an application are any resources which were stored on the clipboard before the application was launched and the "Finder Information". The Finder Information structure is essentially a list of files to be opened or printed by the application. For example, when a user uses shift-clicking to select several documents and an application in the Finder, the names of those documents selected by the user are passed to the application in the Finder Info structure.

There are no standards for passing any other type of information from the launch environment to the application. Some development systems and other systems of several applications use "chaining" instead of "launching" [Inside Macintosh v. II, ch. 2], which allows resources and other data to be passed in the application heap. Unfortunately this requires knowledge of coding details of the applications, and will not be suitable for integrating an arbitrary application with the MPW shell. Applications have been designed under an implicit assumption that all information about operational modes was to be provided interactively by a human user. This leads to the most common impediment in using applications as MPW commands — it is very rare that applications can complete their job without some human intervention (at the very minimum it is usually necessary for the user to select "Quit" from the application menu to return to the shell). This is unfortunate, because the function and mode of applications such as compilers may be fully defined at the time they are invoked, and the user intervention becomes an unnecessary redundancy.

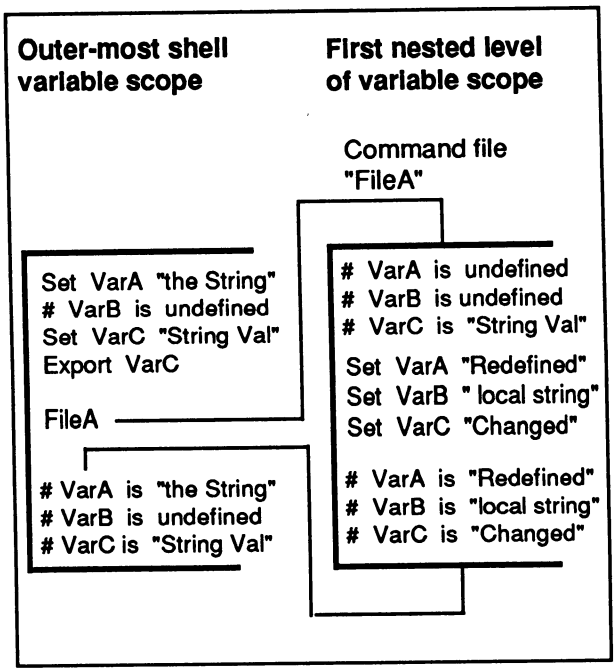


Figure 2. The ins and outs of MPW shell variable scope.

In spite of the inconsistency between the highly interactive user interface of applications and the more mechanized programmable MPW shell, it is possible to develop procedures to integrate shell commands and applications (particularly development systems) with reasonable success. While it is not possible to eliminate human intervention in the applications, we can minimize the actions which the user must perform. Some of the tips in this article might be viewed as an *ad hoc* system of "MPW interface guidelines" to this end.

Applications and MPW

As mentioned above, the MPW shell allows us to launch an application by invoking its name as a command. MPW maintains a shell variable called {Commands} with a list of directories (folders) to search for commands. If the application resides somewhere in this command search path list, merely the filename of the application is sufficient as a command. Otherwise the full pathname, with the volume and all subdirectories leading to the application's resource file must be used as the command name.

The MPW shell provides the standard mechanisms for putting resources on the clipboard or retrieving them from the

clipboard. So passing data to or from an application on the clipboard can be performed without any special considerations. The only other information an application might expect from its launch environment is a list of files in the Finder Information structure. The MPW shell has provided a standard method to pass this information. Filenames which are to be passed in the Finder Information to the application are simply listed as arguments on the command line following the application name. Thus, we can fully configure the standard application launch environment using mechanisms explicitly provided by the shell.

Example 1. Launching application with Finder Info.

There may be a number of applications which are commonly launched by the user from within the shell; this will be especially true if the user has completely replaced the Finder with the MPW shell. I have found it convenient to add a "Run" menu to the MPW shell, using the shell command "AddMenu". This allows selection of my most commonly used applications via menu choices. Such a menu is illustrated in Figure 1.

Most of the items in this menu are straightforward; selecting the menu item simply launches the application because the command field of the "AddMenu" command consisted only of the filename of the application. The last item on the menu, "Word-Paint" is more interesting because it launches a Switcher set of MS Word and SuperPaint. This menu selection launches the application Switcher and passes a Switcher document in the Finder Information. The command line to perform this appears as an argument in the "AddMenu" command which created the menu item:

```
AddMenu Run Word-Paint ∂
'XP40:Word:Switcher XP40:Word:Word∂+SuperPaint'
```

The word AddMenu is the command which will add this item to a menu in the MPW shell. Three arguments follow the command name. The first argument is the word "Run" which is the name of the menu we wish to affect. The second argument "Word-Paint" is the text of the item being added to the menu. The curly-d (∂) is followed immediately by a return and signifies that the command continues on the following line. The third argument is the text appearing between single quote marks. This is the command which will be executed when the menu choice is selected. The command consists of two words separated by a space. The first is the full pathname of the application to launch (Switcher) and the second is the full pathname of the document to pass to the application in the Finder Information. The document name in the Word folder is "Word+SuperPaint"; the extra curly-d is a literal quote of the following character, "+". This quote is necessary because the unquoted "+" symbol is used by the shell as a metacharacter implying certain filename matching rules.

When we select this menu item, the MPW shell closes and Switcher is launched. Switcher then automatically opens the document "Word+SuperPaint" which in turn tells Switcher to launch those two applications. When we quit from both applications and the Switcher, the MPW shell will be restarted.

Of shell variables and search paths

A MPW shell variable is a unique symbolic name that represents storage for a string of characters. Shell commands are available to set the value of a variable, test values, and shell syntax allows the value of a variable to serve as text in a command simply by referring to the variable name. The "Evaluate" command also allows arithmetic operations on strings which can be interpreted as integers. Variables in the shell are entirely analogous to variables in other programming languages. The following examples will use shell variables extensively, so a very brief review of the MPW rules on variables will be helpful.

Values are assigned to variables by the "Set" command:

```
Set  VarName  "the String value"
```

The value of a shell variable is substituted in a command line by enclosing the variable name in curly braces, {}. Thus, when we follow the Set command above with the command:

```
Echo  {VarName}
```

the output printed is:

```
the String value
```

A variable is defined when its name is first encountered by the shell. The Set command can be used to assign a value to the variable; if the first occurrence of the variable is in a command other than "Set", the null string is used as a default value. The only exceptions to this are some predefined variables which are created by the shell when it is launched [MPW Reference Manual, Table 3-2].

The scope of variables is an important consideration in designing shell scripts. Normally, the value of a variable is maintained only within the command file where that variable is defined. For the purposes of this discussion, we include commands typed by the user in windows created by the shell as the outermost scope. When a command file is executed by invoking its filename as a command, a new nested level of variable scope is created to correspond to the command. Variables defined at the level where the command file is invoked are not defined within the inner executing command file. Conversely, variables defined within an executing command file are no longer defined when that command file terminates and control returns to the level of invocation. Variables VarA and VarB in Figure 2 offer examples of these scope rules.

Two important commands modify these rules of variable scope. Using the "Execute" command to run a command script file, instead of using the filename as a command, causes the commands in the file to operate within the current scope rather than creating a new level of variable scope. The "Export" command causes a variable and its value to become available to script files which are nested within the level containing the Export command. Using Export to pass the values of variables into nested command scripts is analogous to passing subroutine arguments "by value" in languages such as C and Pascal. Changes in value of exported variables are therefore not passed back out to the enclosing command file. The variable VarC shown in Figure 2 illustrates the behavior of an exported variable in a shell script. Another somewhat anomalous problem of variable scope occurs when applications are started from shell

scripts, but I leave further discussion of this to one of the examples below.

Standard search paths for files

Many programs need to be able to find and use auxiliary files to process their input and generate their output. For example, compilers often need to be able to merge "include" files containing system definitions with the current source file, and linkers need to be able to find object code libraries when constructing programs. Searching a large hierarchical file system for these auxiliary files may take a prohibitive amount of time, so it is conventional to supply a "search path" — a directory or a small list of directories where the desired files are most likely to be found.

The MPW development systems use shell variables to store search paths for various items. The MPW variables are listed in the top section of Table 1 (and are also discussed in the MPW Reference Manual). Some of the variables (marked with "S" in the table) contain the full pathname of a single directory to be searched; other variables (marked "L" in the table) contain a list of full pathnames of several directories, with the pathnames separated by commas. Using this system, changes of the directory structure or locations of files is accommodated simply by adjusting the values of these variables.

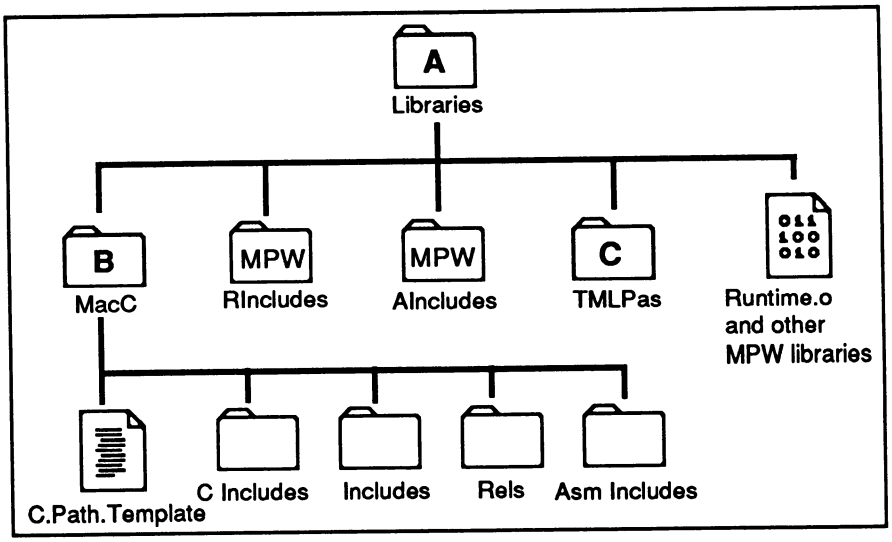


Figure 3. Sample directory arrangement for development systems. Pathname to folder A is maintained in shell variable {Libraries}; to folder B in {MacCLib}; to folder C in {TMLPasLib}.

Other development systems, such as TML Pascal and Consulair MacC, also need to address this "search path" problem. These systems use special resource files in the System Folder to save the search directories they need, and provide software allowing the user to modify the stored paths. In the rest of this section, we try to integrate these two methods of specifying search paths. We want to follow the MPW standard to save search paths in shell variables (Table 1, lower part) and then use these variables in shell scripts which will create the search path resources expected by the applications. Figure 3 shows a directory pattern for libraries, and indicates what folders are listed in various shell variables from Table 1.

Example 2. TML Search Paths.

When TML Pascal or Linker need to find include files, compiled symbol tables for "Uses" or object file libraries, they first look in the directory containing the source file being processed. If the file is not found there, up to five other directories are searched. The full pathname of each of these five directories is saved as a resource (type 'STR ') in the file "Paths File" in the System Folder. The TML package includes a desk accessory which provides a dialog where the user types directory pathnames for the "Paths File".

Following the MPW model, the desired search paths are saved as the values of shell variables {TMLPasLib} and {TMLPasInc}. We must now write the pathname strings from these variables as the resources in the "Paths File". The obvious way to do this is to use the pathname text from the variables to generate an input text file for Rez, the MPW resource compiler; Rez, in turn, will create the 'STR ' resources. Shell script file "MakePPath", in the listings following the article, accomplishes this.

MakePPath opens two temporary windows for use as scratch files. Into one window, the pathname list from shell variable {TMLPasInc} is written, so that it can be parsed with shell editor commands. The second window, TMLPathOut, is where we create the input for Rez. In MakePPath, we write the HFS path from variable {TMLPasLib} into string resource 128, and the first four HFS paths from {TMLPasInc} are written into resources 129 - 132. If {TMLPasInc} contains more than four, only the first four are used.

Table 1. Shell variables for file search paths.

System	Var Name	List *	Purpose
MPW	Libraries	S	MPW runtime libraries
	AIncludes	L	MPW Assembler header files
	RIncludes	S	Type definitions for MPW Rez
	CLibraries	S	MPW C object libraries
	CIncludes	L	MPW C header files
	PLibraries	S	MPW Pascal object libraries
	PIncludes	L	MPW Pascal include files
non-MPW	MacCLib	S	Folder containing Consulair objects, headers in sub folders
	TMLPasLib	S	TML Pascal object libraries
	TMLPasInc	L	more TML object libraries and includes

* S means variable contains only one directory to search
L means variable contains comma-separated list

The main novelty in MakePPath is the loop which breaks the comma-separated list of paths from {TMLPasInc} into individual paths using the MPW editing and selection commands; this could also be useful in breaking down other lists in MPW variables. Let's consider the editor commands which do this. It is assumed that 1)the list of paths has been written as a single line in the file named Tmpfil, and 2)none of the directory names in any of the paths contain a comma.

Before entering the loop, we move the selection in the Tmpfil window (the insertion point) to the beginning of the file, before the first character.

```
Find • Tmpfil
```

Then we loop until we find all the paths in the list. The first selection we make inside the loop is:

```
Find §Δ:Δ/,/ Tmpfil
```

This selects text from the character after the previous selection to the character preceding the next comma. This selection, if it exists, is a valid path, and we can use the notation Tmpfil.§ to refer to this selection. After we use this selection, the command:

```
Find §Δ:§Δ!1 Tmpfil
```

selects the comma following the interesting text. This way when the loop repeats, our next string will not begin with an extraneous comma. If the selection at the beginning of the loop could not be made, then we have reached the last path (which of course will not end with a comma) or there was no text in the variable to begin with. The command:

```
Find §Δ:∞ Tmpfil
```

selects all the text to the end of the file and so selects the last path, if it exists. Since all commands in the shell, including Find, return a status value, we are easily able to test whether each of these selections is successful.

The script file MakePPath is a little more elaborate because it must use only the first four strings from the variable. Also it must write null strings into the 'STR ' resources when fewer than four are specified. After MakePPath has run, the 'STR ' resources it created must still be turned into paths used by the system; the Set Paths desk accessory must be used. When the DA starts, the search paths specified by shell variable are already listed in the DA's dialog box. The user simply needs to select the "Set" button in the dialog to establish these search paths. The MPW script file is helpful because the version of the DA I have doesn't recognize Edit menu operations like Cut and Paste. A sometimes useful/sometimes troubling side effect of Set Paths is that most other applications which search the System Folder for help files or other support data will also search the Set Paths directories for those files.

Example 3. Consulair's Path Manager application.

The Consulair method for specifying search paths is considerably more complicated, flexible and robust than TML's. There are many classes of files which may be sought in different directories. Consulair provides an application, Path Manager, which takes a user-written text file as input and writes a resource file containing search path information for the applications. The input source is a list of any number of search paths for the various classes of target file. Consulair has not published the format of their path resource, so rather than using Rez, we will use MPW variables holding search paths for libraries and includes to augment a Path Manager input file.

Many of the paths used with the Consulair applications are "source relative", meaning that the directories to be searched are specified by partial pathnames appended to the directory where the current source file resides, rather than by full pathnames. This is a strong point in the design of the Consulair Path Manager strategy, and because many paths are source relative, many specifications in the Path Manager input will not change when the directory structure changes. The only paths we really need to worry about are the same ones considered in the MPW and TML systems: the locations of libraries and system include files. Therefore I prepare a template file for Path Manager input which contains the relative paths exactly as needed and which includes MPW variable references only for absolute pathnames to library objects and includes. When the paths change, I just need to change the values of the variables, expand the variables in the template file, and send this as input to the Path Manager application.

Let's consider two sample lines in this template file which specify some directories to search for header files (included .h files) needed by the C compiler. Path Manager input lines look like:

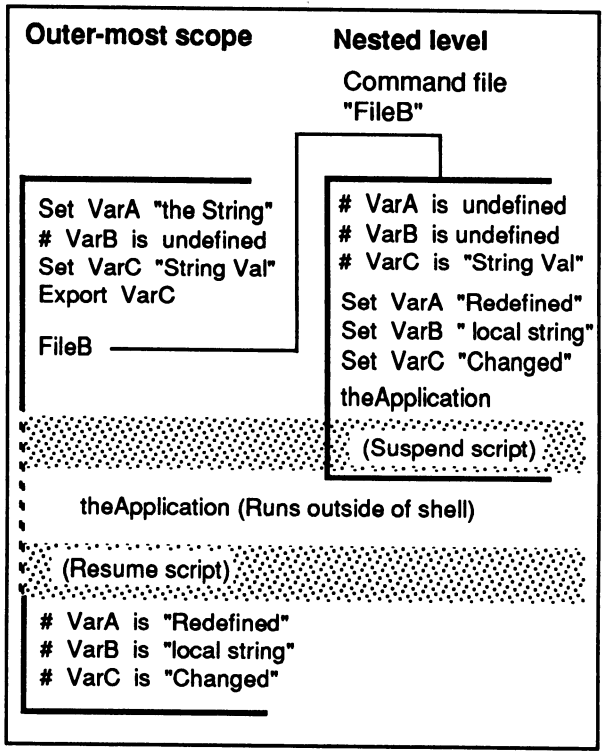


Figure 4. Shell variable scope anomaly associated with the launching of applications. Shell script FileB is implicitly closed.

```
:C_Include *S  *S:Includes:
C_Include  {MacCLib}CIncludes:  {MacCLib}Includes:
```

The first item on each input line ("C_Include" in this example) indicates the class of sought object for the Path Manager and applications to use. A tab–separated list of directories to search follows the class identifier on each line. In the first line of the example, "*S" is a Consulair metanotation which refers to the directory which contains the source file which is being compiled. This input line will cause the C compiler to look in the directory containing the source file and in a folder in that same directory which is named Includes. This input line is interpretable "as is" by the Path Manager and does not need to be changed when different source directories are used. More information and examples of Path Manager input can be found in the Consulair documentation.

The second line of the example fits our present purpose. It lists two directories which will contain system-wide include files. This line uses shell variable {MacCLib} to refer to the two directories as shown in Figure 3. But the Path Manager does not recognize the MPW shell variables, so we must expand the variable in this input line before passing the input file to the application. Shell script MakeCPath accomplishes this task, and launches the Path Manager. Note that the placement of the template file as shown in Figure 3 allows MakeCPath to use variable {MacCLib} to locate the template. If the template isn't found we will know that the variable is not up to date!

MakeCPath uses the template file as input and writes the text file which will be read by the Path Manager application. The file for the Path Manager is named MacC.Path and placed into the System Folder for later reference. Expanding the variables in the template file without altering any other text requires a little bit of mental contortion and the use of an intermediate scratch window. The first step in the process is to open a window to the template file. In the template window, we literalize any characters which might be mistaken for shell metacharacters. For example, the asterisks in the "*S" entries above must be literalized. The sequence of commands

```
Find •
Replace -c ∞ /∂*/ '∂∂∂*'
```

inserts three curly-d's in front of every asterisk in the template. The three adjacent curly-d's cause literal interpretation of the asterisk through the next two shell commands which interpret the line. Note that in general, 2^n–1 curly d's will literalize a character n times. Any other MPW shell special characters which appear in the template must be similarly quoted. MakeCPath only literalizes asterisks and tabs; add your own as needed. Note that the curly-d's are being added directly to the template window. When we are finished with the template, we must close this window with the -n option so the changes won't be written to the file on the disk.

Once the template is prepared, loop to process it one line at a time. MacTutor masochists who read the code listings first may be mystified at the command

```
Echo Echo ∂"'Catenate §'∂" >> "{tmpfile2}"
```

There may be a prettier way to do it, but this does work. This is an echo command which writes another echo command as a line in the scratch window. The name of the scratch window is in the variable {tmpfile2}. The Catenate§ inside grave symbols is replaced in the scratch file by the line currently selected in the template file. This catenate command consumes one level of literalization in the template, so ∂∂∂* in the template becomes ∂* in the scratch file. Shell variables such as {MacCLib} are not expanded in the scratch file.

When the loop finishes, the scratch file contains one echo command corresponding to each line of the original template file. Execute the commands in the scratch file using

```
Execute "{tmpfile2}" >> "{outfile}"
```

and the variable outfile contains the filename MacC.Path which will become the input to the Path Manager application. The echo commands in the scratch file consume one more level of literal quotes (so ∂* becomes * in the output file) and they also expand any shell variables into their values.

MakeCPath contains additional commands which facilitate convenient behavior after the application runs; these features will be considered in the next section. Finally the MakeCPath script launches the Path Manager, passing the file MacC.Path in the Finder Information structure.

Now we run into some of the difficulties associated with applications in shell scripts. Although I passed the input with the Finder Info, the Path Manager (ver. 5 at least) doesn't notice it. I still have to manually select a menu option, and then locate the text file using a StandardGetFile dialog. The best I can do is make this process as painless as possible. In the MakeCPath script, just prior to launching the application, make the System Folder into the default directory using the Directory command. Now when the Path Manager throws up the StandardFile dialog, it will initially be showing the files in the System Folder. Since MakeCPath wrote the file MacC.Path into the System Folder, we won't have to search through the directory hierarchy to find it. When the Path Manager generates the PATH resources, I will also have to manually specify the output file in a StandardPutFile dialog, and then manually select Quit from the menus. We've saved ourselves some work in editing the input file, but if the application just used the Finder Information all the manual interaction could be eliminated.

Altered states

Now let's consider the things which happen in the shell environment when we launch an application. In the absence of Switcher–like trickery, we must terminate any current application prior to launching another one. The MPW Shell is itself an application and so must terminate before launching another and must be relaunched when the external application quits. In the parlance of the MPW Reference, MPW is "suspended", the external application runs, and then MPW is "resumed". The processes necessary to suspend and resume the MPW shell are performed by shell scripts provided with the MPW system, named (appropriately) Suspend and Resume.

The Suspend script saves all currently defined shell variables, exported variables, command aliases, menu items, the

© The Definitive MacTutor, Vol. 4

current working directory, and any open windows. Then when the MPW shell is relaunched, the Resume script uses this saved information to return the shell to nearly the same condition as when it was suspended. There is one important difference between the suspended state and the resumed state, which is tersely documented in the MPW Reference. An application may be launched from a nested command script, such as MakeCPath from the preceeding example, but the shell always resumes at the outermost command level. The shell implicitly exits any nested command scripts when suspended.

This is the variable scope anomaly I to which I earlier alluded. The Suspend script saves the names and values of variables at the current level of scope, but the Resume script sets these variables again at the outermost level (see Fig. 4). WARNING: I use this as a "feature", but future versions of the shell may consider this a "bug"; some of my shell scripts will require minor modification if future releases change this behavior (I am using MPW version 1.0 for the examples of this article). *[Current version is 2.0b1. -Ed]*

If you launch applications within scripts, as I do, some defensive scripting practices are essential. Any outer-level shell variables you want around after Resume must be exported from the outer level and not modified by nested shells. Any inner-level variables you don't want around after Resume must be Unexported and Unset in the script file prior to the command which starts the application. Remember that you only need to worry about this when you are using shell scripts like MakeCPath which launch applications.

Often, launching an application is not the final step in the process, but the shell script which launches the application is not restarted from that point. I get around this by setting up commands to be executed by the Resume process. I put these commands into a file and save the name of this file in a shell variable called {PostApp}. I have modified the Resume script file to include the commands:

```
Execute "{PostApp}"
Set PostApp NullFile
```

When the MPW shell resumes after an application, commands which I saved prior to suspending can be executed. Variable {PostApp} is reset to a default empty command file which is named "NullFile". An alternate method that doesn't depend on the Suspend/Resume shell variable scope anomaly is to save the commands under a specific file name, say "UserResume". Then the Resume script should contain

```
Execute "UserResume"
Echo "" > UserResume  #Deletes contents
```

I prefer the first method (as long as it works) because the command file contents stay available for debugging when I am writing a new command script. Note that in either method the changes to file Resume are permanant, and we merely change variable {PostApp} or file UserResume to fit the needs of the moment.

I return to the MakeCPath script for an example of a useful Resume task. Recall that MakeCPath changed the current working directory to the System Folder just prior to launching the Path Manager. This means that the Suspend script will save this directory and Resume will restore the System Folder as the current working directory. This will probably not be the directory from which the user started the MakeCPath script, so when we Resume, we are not where the user expects. A more user–friendly thing to do is to save the original working directory and restore it at the end of the Resume process. The commands which set this up are found near the beginning of MakeCPath. Variable {PostApp} is given the name of the resume command file "UserResume". The original directory is used in a command stored in file UserResume:

```
Directory "OldDirectoryName "
```

Now when MPW resumes, the user will be in the same place in the directory hierarchy as when MakeCPath was invoked. This technique of adding transient commands when resuming from applications has considerable utility and we will see more extensive examples presently.

MPW Make with Applications

The shell scripts developed in the preceding text, while they do provide instructive examples of technique, are only moderately useful. The directory structure and library file locations shouldn't change too often, and these shell scripts are unlikely to be needed more than a couple of times a year.

Building and maintaining programs, on the other hand, is more of a daily occupation. Within the MPW, these tasks center around the tool "Make", which is an elegant system of rules to emit commands which depend upon the relative ages of files. If we apply what we learned in the previous examples to this problem, we are rewarded by a system of genuine utility — an integration of MacC and TML Pascal systems which the developer accesses from entirely within the MPW environment.

Basics of Make

"Make" programs are widely available utilities found in the MPW, in a variety of Mac shareware and commercial offerings and in most Unix—based systems, among others. The traditional use of a Make utility is to keep programs up to date with a minimum of processing, compiling only when source files change and linking only when relocatable objects change. This behavior is controlled by a text file containing 1)dependency rules specifying which files are constructed from which predecessor files and 2)a list of commands which perform the operations necessary to update the target files. Since F. Alviani's April '87 MacTutor article elaborated some of the sketchy documentation on Make, I won't repeat that detail here. An important point in the adaptation of MPW Make for application—based development is that the full range of legal commands can be generated by a make script. This allows us to generate "smart" sequences of commands which do much more than just compile and link (and "Rez").

The MakeCPath command described earlier provides the simplest possible illustration of Make which does not compile or link. The final output of MakeCPath is a resource file in the System Folder named Paths.Rsrc. This output depends, naturally enough, on the input — the template file, C.Path.Template, and any shell variables , such as {MacCLib}, which are used in the

template file. Dependencies in Make can only be specified with respect to files, not variables, so the values of {MacCLib} and related variables are initialized in the MPW script file "Startup". "Startup" is a script supplied by the MPW system which is executed when the MPW shell is launched from the Finder. If I change the value of variable {MacCLib} by editing this script file, the dependency of Paths.Rsrc on the variable becomes equivalent to a dependency on file "Startup". (I put these variables into "Startup" instead of "UserStartup" because I edit file "Startup" much less frequently).

The input to Make specifies these dependencies in a special notation:

```
Paths.Rsrc  ƒ  Startup  C.Path.Template
    MakeCPath
```

The line containing the *ƒ* symbol is the dependency statement. Files listed to the left of the *ƒ* depend upon (or are made from) files listed to the right of the symbol. Dependent files (left side) need to be updated if they are older than any of the predecessor files (right side). Note that in an actual Make script, you might need to specify the appropriate directories as well; I have omitted this detail for the clarity of the example. The dependency statement may be followed by a list of commands which perform the necessary update. Commands are distinguished in the Make input because each line of a command list begins with whitespace (space or tab characters). In the example above, "MakeCPath" is the command which will update the target, "Paths.Rsrc".

Example 4. The MixMake command

Here I develop a shell script and Make input which work together to build programs. To effectively use applications with shell scripts, we need an understanding of their behavior in this environment. I start this example, therefore, by cataloging what happens to each application when it is launched from the MPW shell with an appropriate input file specified in the Finder Information structure. MacC (ver. 5) does not use the Finder Info properly, and in fact gives an ID 02 bomb (or worse) when you try from the shell. The Consulair linker will process a link script, but when finished puts up a standard file dialog, and requires user interaction to quit to the shell. If the link fails, the Consulair linker will launch Edit, instead of the shell, if it can find it. Removing the Edit application completely from the disk is helpful in using the MixMake shell script. TML Pascal will compile one file from the Finder Info (even if several were listed). After compiling, it requires the user to click OK buttons, to select other files or to quit manually. The TML linker will use a link script passed in the Finder Info, and then runs the program it built and/or returns to the shell. Amazingly, the Microsoft FORTRAN compiler (ugh) has almost ideal performance in this environment; it compiles the file passed in the Finder Info and returns to the MPW shell, and only requires the user click an OK button if a compile fails.

So far, things are looking a bit dim, but utility applications can save the day. The Exec application, provided with the Consulair system, is a simple utility which runs Consulair applications in sequence according to an input text file. Like the Path Manager, when this is launched from the shell the user must select the input file manually, but it generally will return to the shell when done. Furthermore, the Consulair linker and compiler recognize that they have been launched by Exec instead of the shell, and they accept and process input files and return to the Exec without user intervention. The MPW method simply needs to use this launching indirection through Exec to perform an automated build. The TML applications unfortunately do not change their behavior for Exec. Many other free/cheap utilities (for example, Darin Adler's Sequencer) will run applications with input files in sequence and may be more appropriate for the non-Consulair applications.

The MixMake command, which builds programs with MacC and TML involves a significant amount of programming beyond the simple Make utility. Since Make will, however, form the heart of the MixMake command, let's consider first the nature of a MixMake makefile. When the dependency rules indicate that a source must be recompiled or a program relinked, a MixMake makefile generates a line of input for the Exec application instead of a command to run the compiler or linker directly.

A sample makefile for an application is given in the Listings. The default dependency rules, which are applied to any filename with the appropriate filename extension, effectively handle all the compile steps. These default dependency rules may require directory dependency rules as well to operate correctly, but this will tend to vary with each build target. The default compiling rules are:

```
.Rel ƒ .C
    Delete -i {TargDir}{Default}.Rel
    Echo {DepDir}{Default}.C >>{Cfile}
    Set RunMacC 1
.Rel ƒ  .Asm
    Delete -i {TargDir}{Default}.Rel
    Echo {DepDir}{Default}.Asm >>{Cfile}
    Set RunMacC 1
.Rel  ƒ  .Pas
    Delete -i {TargDir}{Default}.Rel
    Echo 'Pascal  {DepDir}{Default}.Pas  Exec'∂
        >>{Efile}
    Set RunTPas 1
.Rsrc  ƒ  .R
    Delete -i {TargDir}{Default}.Rsrc
    Rez {RezOpt} -o {TargDir}{Default}.Rsrc ∂
    {DepDir}{Default}.R
```

Variable {Efile} is the file which is being prepared for Exec input. Variable {Cfile} is a filename with extension ".Files" which contains a list of sources to be compiled by the MacC application. Setting the variable {RunMacC} tells the enclosing MixMake shell command to add a single command to the Exec input file which will launch the MacC compiler with the ".Files" file. If resources are to be compiled with Rez, the commands are generated by Make and executed directly by the MixMake script. This arrangement assumes that non-CODE resources are compiled by Rez but the application is assembled from CODE and these resources by a subsequent link step. When a Pascal source

is to be compiled, the makefile appends an appropriate command to the Exec input file. Since the Pascal compiler does not interface well with Exec, the user must select (and know in the first place) all the sources to be compiled in Standard File dialogs. When the user selects Quit from Pascal, the Exec processing is terminated and MPW resumes. To complete any MacC compiling and linking, the user would have to run the MixMake command a second time.

A much nicer method uses Adler's Sequencer and could be adapted most to any application-running utility; a Pascal compiler build rule is implemented as Rez input which is used to produce a Sequencer which will build the desired Pascal files. The user does not need to know which Pascal sources are to be rebuilt; the user just selects Quit in Pascal's file menu after each compile, and the next out-of-date file is compiled automatically. The final operation by the Sequencer after all Pascal files are compiled is to launch the Exec to handle C compiling and Linking. I have included both versions of MixMake with the source code, but will not describe this one further in the text. The extra effort to use the Sequencer (or similar utility) has significant benefit. The Suspend/Resume scripts take a noticeable time to execute, so we want to do as much work as possible outside the shell for each Suspend/Resume cycle.

The sample Makefile.Template in the listings illustrates the appropriate specifications needed to build a particular file. It adds a directory dependency rule before the default compiler rules. A dependency rule for the link step must explicitly list the link script (".Link" file), relocatable object files (".Rel" files) and compiled resources (".Rsrc" files) which are needed. The command generated by this dependency rule is again an appropriate line of Exec input. Particular care must be taken to prevent building a program with some up-to-date object files and some which are not up-to-date because of compiler errors. The delete command in the compiler default rules described above is executed by the MixMake script and prevents a successful link until the object is successfully rebuilt by the compiler. The MacC compiler rules don't need to delete obsolete objects because the Exec-MacC system allows a time saving alternative. Each line of Exec input consists of four fields: 1)the application to launch, 2)the input file for the application to process, 3)an application to run if the application in field 1 runs successfully (normally Exec to continue processing), and 4)an application to run if the application in field 1 fails. The line of Exec input which MixMake uses is

```
C input.Files Exec Done
```

The application "Done" is a trivial program containing only an ExitToShell command. If any of the files to be compiled or assembled by MacC have errors, the Exec processing terminates before any attempt to link, and the Done program Resumes the MPW shell instead.

It is important not to list the MPW Shell directly as an application in the Exec input. The MPW shell could be *launched* by Exec, but then selecting Quit from the MPW File menu *resumes* the MPW shell instead of starting the Finder. This is quite confusing, so we don't do it.

There are few restrictions on MixMake. The MixMake command shell takes user input arguments exactly like Make, and uses a makefile with the appropriate default rules described above to generate an input file (set up to run in the appropriate order) for the Exec application. The Make tool's options –d, –e, –f can be used in arguments for MixMake. Make options –p, –r, –s, –t, –u should not be passed to MixMake; they may not behave as expected. Options –r, –s and –u can be used by invoking Make instead of MixMake and using the MixMake input dependency file. The key to adapting the sample makescript to a particular purpose is in the list of predecessors (usually for a link rule) and directory dependency rules. It is important to follow the advice in the MPW Reference and assure that directories are specified in default directory dependency rules in precisely the same form (full or partial pathnames) as the corresponding explicit predecessor files. These should also be consistent with the search paths of the applications. The version of MixMake which uses the Sequencer requires that full paths are specified for all directories.

In the simplest implementation, MixMake would simply launch the Exec or another application runner. A more sophisticated MixMake, however, provides Resume tasks for "nice" error handling when the build steps don't succeed. I'll finish up the article with a description of commands which help with analyzing diagnostics.

OOPS! What about those errors?

All of the Consulair and TML applications write diagnostic files when compiling or linking fails. A convenient feature to add to the mixed system is a facility to automatically open a bad source listing its corresponding diagnostic file simultaneously. Shell script "CompErr" is just such a "diagnostic viewer", running sequentially through all bad .C and .Asm sources, then all bad .Pas sources, and finally any failed linker scripts. The information needed by CompErr to tell which are the "bad" sources from the most recent build attempt consists of several files and shell variables. These files and variables are provided by a Resume script which is installed by the MixMake command. A copy of the commands MixMake generates as a Resume task is kept in file "MixMakeResume". These Resume tasks and the CompErr command are more limited than MixMake itself, and generally work only when all the necessary sources and diagnostics are located in a single directory and if that directory is the current working directory when the MixMake command is invoked. This restriction has not been onerous for me, but if necessary more elaborate Resume and CompErr scripts could circumvent the limitation.

The MixMake Resume commands write files which contain lists of paired source and diagnostic files. The Resume commands also install an "Open Diagnostics" item in the MPW shell File menu that runs the "CompErr" command. The files summarizing the diagnostic file names are scanned sequentially by "CompErr"; each time the menu item is selected, the next pair of source and diagnostic files is opened for the user to view and edit. When all the diagnostic files have been opened, the menu item deletes itself.

The MacC compiler creates a file with the basename of its .Files input and the name extension .Fer, in which it writes a list

of files which failed to compile and their corresponding diagnostic files. For Pascal compiler and linker errors, the Resume task needs to explicitly search the directory to create such a list. For example, in a directory containing a Pascal source with extension .Pas, the existence of the same name with extension .Per suggests a failed compilation. Existence of the diagnostic file is not sufficient evidence however; the source might have been corrected since the diagnostic file was created. The Resume task only lists source files which are out-of-date with respect to their corresponding diagnostic files. The Make tool, a real workhorse, assures this. Here are dependency rules for finding bad sources from diagnostic files:

```
.Pas  ƒ  .Perr
   Echo {DepDir}{Default}.Pas ∂
   {DepDir}{Default}.Per >>TMLP.Fer
.Link  ƒ  .Lerr
   Echo {DepDir}{Default}.Link ∂
   {DepDir}{Default}.Lerr >>TMLP.Fer
```

All the Pascal diagnostics are listed in a file named TMLP.Fer and all the link diagnostics in a file named Link.Fer. The dependency rules above are in a makefile permanantly stored (never accessed by the user). This has the advantage of allowing a general command to be built around it, but the disadvantage that it will only work within the current directory.

CompErr uses these files (from the most recent MixMake build only), opening one source/diagnostic pair for each invocation. CompErr should always be accessed through the "Open Diagnostics..." menu item ("Dial Cmd–E for Errors") because the menu procedure moves to the directory where the build occurred, opens the files, then returns the user to the current directory. Since CompErr works sequentially through the .Fer files, we need to save the current position in the files after each invocation. Counting line numbers might work, but I have chosen to use the method of saving the current window selection with the file.

In MPW, files and windows into those files are distinct, although intimately related. When we save a window, the selection is saved too. After the window is closed, the saved selection reappears when the window is opened. The Save command is pretty smart, and only writes changes to the disk when the text has changed, not when only the selection changes. The CompErr script forces saving new selections by writing innocuous white space into the .Fer file after each change of selection. The marker for where CompErr is in the file is therefore saved elegantly in the file itself.

OK, let's finish up with an ugly scenario. It is Friday afternoon, you are about to leave on vacation and a lengthy build has just resulted in 30 or 40 erroneous source files. You don't particularly want to re-Make the whole shebang just to prime CompErr when you return. NO PROBLEM! If we can have Suspend/Resume tasks, we can also have Quit/Startup tasks.

Two scripts delivered with the MPW system, appropriately enough named "Quit" and Startup, handle special tasks when the user elects to terminate the MPW shell by quitting to the Finder and when the shell is launched from the Finder. I have added routines to Quit and Startup which allow me to preserve any state variables I want between invocations of the shell. In the Quit script, I execute a command file named "QuitPermVars". In QuitPermVars are commands which save two exported state variables required by CompErr: {MakeErr} and {CompErrDir}. QuitPermVars writes Set commands to reproduce these two variables, corresponding Export commands, and an AddMenu command to install the "Open Diagnostics" item. These commands are saved in a file named "StartPermVars". Then the Startup script executes StartPermVars when the shell is launched from the Finder. So you can leave your problems behind, and when you come back with clear and refreshed mind, CompErr will dutifully pitch them out to be solved.

Conclusions

Although it may look a bit ungainly because of the number of small command files, the system I have presented here as an example is quite workable and I actually use it every day. Programming the shell is very challenging compared to other programming because of the global nature of many of the manipulated data and the resulting high data connectivity of "independent" modules. But the rewards of mastering shell programming can be great. Before I attempted this project, in a fit of frustration, I had presumed it would be impossible to integrate these diverse applications with the shell. The flexibility of the MPW should not be underestimated however, and this collection of command scripts provides me with the same level of performance from the development applications as when they stand alone plus an improved text handling and resource handling facility.

I should add that the systems integration designed into the Consulair C applications contributes a greatly to the effectiveness of my scheme. The behavior of the various applications (Consulair's, TML's and others I've tried) could have been better for use from the shell however. A few simple considerations can insure smooth integration of applications and MPW shell.

Making applications MPW friendly

First and foremost, applications which may be launched from the MPW shell (or other non-Finder shells, like A-UX?) should handle the Finder Information structure. No assumptions should be based on Finder behavior (for example, assuming that documents and application must reside in the same folder when they have different file creator words).

Secondly, applications with strongly modal behavior should probably have the ability to read an input file to set those modes and perform its action. An application which can do its job with just one or two mouse clicks by the user in menus or dialog boxes has strongly modal behavior, and compilers are an excellent example. There is unfortunately no way to establish a standard, but the following suggestion should provide a sufficiently convenient interface with the MPW shell. If you have a strongly modal application, consider using a TEXT file, perhaps with a distinctive name or filename extension, as an optional way to set the modes and actions to be performed. And finally, when an application can be controlled modally in this way, provide an option to Quit or Transfer without user intervention. Yes, I know this flies in the face of the User Interface Guidelines, but the

example of this article shows that the User Interface is not always the Best Interface.

A challenge to future MacTutor authors

The MPW system provides a Help tool which provides on-line documentation for shell commands. In the listings for this article I have included a script named Man (for Manual) which extends the help procedure to search both the default help file and a second help file named Local.Help. The Man procedure also uses the Canonical spelling tool to convert your standard command aliases to the command names known to the system. Just put an alias and corresponding command on a line in file Alias.Dict.

With the Man command, I have provided a Local.help file with entries for the command scripts developed in this article. Hopefully the command scripts and tools in future MacTutor articles can include relevant help file entries with their source code contributions.

Listings

```
###         file ReadMe
###     ReadMe - instructions for installing
###     MixMake and related commands in your
###     MPW system.
###         W.G. Powell 1987

###     Because parts of the MPW system will usually
###     be scattered throughout a number of HFS
###     folders (directories) the following guide
###     will be helpful in adding the sources
###     provided here. You may need to customize file
###     paths in some of the commands to match your
###     directory arrangement.

###     Into the MPW directory (the folder containing
###     the MPW Shell application itself, and referred
###     to by shell variable {MPW}, place the files
###     contained in the source disk folder "for MPW folder"
###         Local.Help
###         QuitPermVars
###         StartPermVars
###         :MakeSupport:  the folder and its contents:
###                 MakePErrs.mk
###                 Mixmake.Template.1
###                 Mixmake.Template.2
###                 MixMakeResume
###                 Paths.mk

###     Into the Tools directory (or any directory listed
###     in the MPW command search path (Shell variable
###     {Commands} ), place the files contained
###     in the source disk folder "for Tools folder"
###         Alias.Dict
###         CompErr
###         MakeCPath
###         MakePPath
###         Man
###         'MixMake ver.1'
###         'MixMake ver.2'
###         NullFile
###         UpdatePaths
###     If you use Darin Adler's Sequencer with MixMake,
###     rename version 2 "MixMake".
###     If you use Consulair/MDS Exec only with MixMake,
###     rename version 1 "MixMake".
###     Use the appropriate template in directory
###     {MPW}MakeSupport: as a guide in preparing
###     your own MixMake makefiles.

###     Into the Applications directory (or any directory
###     listed in the MPW command search path
###     (Shell variable {Commands} ), place the applications
###     you will be using with the system.
###         Consulair applications:
###             C
###             Exec
###             Link
###             'Path Manager'
###         TML applications:
###             Pascal
###             'TML Link'
###         Other applications
###             Sequencer *if used
###             Done * supplied with this disk
###     Note that if both TML and Consulair
###     Linkers are present, one must be
###     renamed.
###     Also, since TML Desk Acc. "Set Paths"
###     is rarely used, I install it in the
###     MPW Shell application, instead of a
###     precious System File slot.

###     The following changes should be made to the
###     scripts supplied with the MPW shell

##################################################

### Add these commands to file Startup
###     Modify to match your directory hierarchy.

#   {PostApp} - name of command to execute when
# MPW is RESUMED, after running a Finder application (WGP)
            Set PostApp "NullFile"
            Export PostApp

#   {MacCLib}is directory for MacC includes and Libs
            Set MacCLib {Libraries}MacC:
            Export MacCLib

#   {TMLPasLib} is directory for TML Pascal interfaces, libs
    Set TMLPasLib "{Libraries}TMLPas:Pascal System TML:"
            Export TMLPasLib

#   {TMLPasInc} - Directories to search for
#               TML Pascal interface files
            Set TMLPasInc "{Libraries}TMLPas:"
            Export TMLPasInc

##################################################
###     Add these commands to file UserStartup
###     Modify to match your directory hierarchy.

# Recover values of saved state variables
        Execute "{MPW}StartPermVars"

##################################################
###     Add these commands to file Quit
###     Modify to match your directory hierarchy.

###     Values of variables below are saved
###     between invocations of the shell
            Execute "{MPW}QuitPermVars"

##################################################
###     Add these commands to the end of file Resume
###     Modify to match your directory hierarchy.

#   System for nice return from applications
#   The shell variable "PostApp" contains a command to
#   Execute when resuming.  This variable is reset to a
#   default so unexpected results are avoided when another
```

```
#   application is run.
        Execute "{PostApp}"
        Set PostApp "NullFile"

##########################################################


###       File MakePPath
# Try using Rez to write path info to the TML path file
# in the system folder.  Basically, just need to expand the
# TMLPasLib and TMLPasInc vars and write some Rez input
# with it.
#       W.G. Powell 1987  for MacTutor

# Nonzero command status doesn't mean errors,
# so don't terminate
Set __oldExit {Exit}
Set Exit 0
# Open temporary windows
Open -n -t Tmpfil
Open -n -t TMLPathOut
# Write Rez input using shell's TML path variables
Echo "#include ∂"Types.r∂"">TMLPathOut
# Create STR resource for single path in {TMLPasLib}
echo "resource ∂'STR ∂' (128) ∂( ">> TMLPathOut
echo "∂"{TMLPasLib}∂"">> TMLPathOut
echo "∂);">> TMLPathOut
# Put list of paths from {TMLPasInc} into a window
echo {TMLPasInc} > Tmpfil
# Break comma-separated list into individual paths
Find o Tmpfil
Set IDno 129
Loop
    If ({IDno} < 133)
        Find §∆:∆/,/ Tmpfil
        If ({Status} == 0)
        echo "resource ∂'STR ∂' ∂({IDno}∂) ∂("">>TMLPathOut
            echo "∂"`Catenate Tmpfil.§`∂"">>TMLPathOut
            echo "∂);">>TMLPathOut
            Set IDno 'Evaluate ({IDno} + 1)'
            Find §∆:§∆!1 Tmpfil
        Else
            Find §∆:* Tmpfil
            If ({Status} == 0)
        echo "resource ∂'STR ∂' ∂({IDno}∂) ∂("">>TMLPathOut
            echo "∂"`Catenate Tmpfil.§`∂"">>TMLPathOut
                echo "∂);">>TMLPathOut
                Set IDno 'Evaluate ({IDno} + 1)'
                Loop
                    If ({IDno} < 133)
            Echo "resource ∂'STR ∂' ∂({IDno}∂) ∂(∂"∂"∂);"∂
                        >>TMLPathOut
                        Set IDno 'Evaluate ( {IDno} + 1)'
                    Else
                        Break
                    End
                End
            Else
                Loop
                    If ({IDno} < 133)
            Echo "resource ∂'STR ∂' ∂({IDno}∂) ∂(∂"∂"∂);"∂
                        >>TMLPathOut
                        Set IDno 'Evaluate ( {IDno} + 1)'
                    Else
                        Break
                    End
                End
            End
            Break
        End
    Else
        Break
    End
End
```

```
# Use Rez to write the Paths file.
# Close temporary windows.
Close -n Tmpfil
Rez -o "{SystemFolder}Paths File" -t 'ZSYS' -c 'MACS' ∂
                                        TMLPathOut
Close -n TMLPathOut
Set Exit {__oldExit}
Unset __oldExit
Exit 0
##############################################


###          File MakeCPath
###       MakeCPath - MPW Shell Script
### Set up input for Consulair Path Manager
### using appropriate MPW shell variables.
### William G. Powell 1987 for MacTutor

#    Save current working directory
#    and return to it after Path Manager quits.
Set PostApp ∂""{MPW}"UserResume∂"
Echo Directory ∂"'Directory'∂"> "{MPW}UserResume"
Echo "Delete -i MakePtOut">> "{MPW}UserResume"
Echo Unset outfile >> "{MPW}UserResume"
#    Specify the input template file
Set infile "{MacCLib}C.Path.Template"
#    Place the output text file (input for Path Manager)
#    into the system folder where easily found
Set outfile "{SystemFolder}MacC.Path"
#    Assign temporary file to window and clear it.
Set tmpfile2 "{MPW}tmpfile2"
Open -t -n "{tmpfile2}"
Clear o:* "{tmpfile2}"
#    Open the template window
Set _oldExit {Exit}
Set Exit 0
Open -t "{infile}" ≥≥ Dev:Null
If {Status} != 0
    Alert "Cannot Find/Open the input template file.∂n∂
Check shell variable MacCLib, currently∂n{MacCLib}"
    Close -n "{tmpfile2}"
    Set PostApp "NullFile"# Reset default Resume tasks
    Exit 2
End
Set Exit {_oldExit}
Unset _oldExit
#    Add necessary quote characters to the template file.
Find o
Replace -c * /    / '∂∂∂    ' # tab characters
Find o
Replace -c * /∂*/ '∂∂∂*'
#    Loop to copy each line of input,
#    expanding variables where necessary.
Set loopend 'count -l "{infile}"'
Set loopcnt 1
Loop
    If {loopcnt} > {loopend}
        break
    End
    Find {loopcnt}          # Select next line of input
    # Write command into temporary window which will
    # expand shell variable paths when writing to {outfile}
    Echo Echo ∂"'Catenate §'∂">> "{tmpfile2}"
    Set loopcnt 'Evaluate ( {loopcnt} + 1 ) '# increment
End
# Don't terminate script file on errors below - the possible
# errors are not important enough!
Set _oldExit {Exit}
Set Exit 0
# Open, clear, and write the output file
Open -t -n "{outfile}"
Clear o:* "{outfile}"
Execute "{tmpfile2}" >> "{outfile}"
# Close all working windows, saving only the output.
#Close -n "{tmpfile2}"
```


© The Definitive MacTutor, Vol. 4

```
Close -n "{infile}"
Close -y "{outfile}"
# Remove all unneeded variables.
Unset infile
Unset tmpfile2
Unset loopend
Unset loopcnt
# Change "exit on error" flag back to original value
Set Exit {_oldExit}
Unset _oldExit

# Change default directory to System Folder
Directory "{SystemFolder}"
# Execute the Path Manager, passing our new file
# in the Finder Information structure.
"Path Manager" "{outfile}"

Exit 0  # Control should never reach this statement
##########################################

###      File Paths.mk
# Makefile to ensure that Consulair and TML search paths are
# up to date with appropriate shell variables

#            W. Powell 1987 for MacTutor

#    Variables containing search paths for library files and
#    include files are defined in the Startup script so we
#    want to rebuild when the Startup file is changed.

### TML Pascal Paths File
"{SystemFolder}Paths File" ƒ    "{MPW}Startup"
    MakePPath

### Consulair MacC Path.Rsrc File
"{SystemFolder}Paths.Rsrc" ƒ    "{MPW}Startup" ∂
                "{MacCLib}C.Path.Template"
    MakeCPath
##############################################

###      File MixMake ver.1
### MixMake version 1.
### Uses only Consulair Exec application runner
### Script file to build programs with
### MacC or TML systems
###      W.G. Powell   1987 for MacTutor

### Initialize variables
#    Exec and MacC input and scratch files
Set Efile "Exec.Job"
Set Cfile "MacC.Files"
Set Lfile "Link.Tmp"
Delete -i "{Efile}" "{Cfile}" "{Lfile}"
#    Application run flags
Set RunMacC 0
Set RunLink 0 # Consulair
Set RunTMLLink 0
#    Make's working directory
Set MakeWD "`Directory`"
Export MakeWD

### Run the Make tool
make ("Parameters") > MakeOutApps ≥ Dev:StdOut

### Construct the input for the Exec application
Execute MakeOutApps
Delete MakeOutApps
If ("{RunMacC}" == 1)
    Echo "C  {Cfile}  Exec Done">>"{Efile}"
End
If ("{RunLink}" == 1 || "{RunTMLLink}" == 1)
    Catenate "{Lfile}" >>"{Efile}"
    Delete -i "{Lfile}"
End
Unset RunMacC
Unset RunLink
Unset RunTMLLink
Unset Lfile
Unset Cfile
Unset MakeWD

### Set up the Resume tasks for after the applications
### Primarily for error handling
Set PostApp "{MPW}UserResume"
Echo "Unset Efile" >"{MPW}UserResume"
Catenate  "{MPW}MakeSupport:MixMakeResume" >>"{MPW}UserResume"

### If Diagnostic menu item is still around from
###  an earlier build cycle, delete it.
Set __oldExit {Exit}
Set Exit 0
DeleteMenu File "Open Diagnostics…" ≥ Dev:Null
Set Exit {__oldExit}
Unset __oldExit

### Run applications using Consulair's Exec
Exec "{Efile}"
##########################################

###      File MixMake ver.2
###      MixMake version 2.
#### Uses Sequencer application and Consulair Exec

### Script file to build programs with
### MacC or TML systems
###      W.G. Powell   1987 for MacTutor

### Initialize variables
#    Exec and MacC input and scratch files
Set Efile "MixMake.Job"
Set Cfile "MacC.Files"
Set Lfile "CLink.Tmp"
Set  Tfile "TLink.Tmp"
Set Seqfile "Sequencer.R"
Delete -i "{Efile}" "{Cfile}" "{Lfile}" "{Tfile}"
Echo "#include ∂"Types.r∂"">"{Seqfile}"
Echo "include ∂"{MPW}Applications:Sequencer∂" ∂'CODE∂' ;"∂
                        >>"{Seqfile}"
Echo "resource 'STR#' (128,∂"MixMake Compile commands∂")
∂(">>"{Seqfile}"
Echo -n "∂( ">>"{Seqfile}"
#    Application run flags
Set RunMacC 0
Set RunLink 0 # Consulair
Set RunTMLLink 0
#    Make's working directory
Set MakeWD "`Directory`"
Export MakeWD

### Run the Make tool
make ("Parameters") > MakeOutApps ≥Dev:StdOut

### Construct the input for the Exec application
Execute MakeOutApps
Delete MakeOutApps
If ("{RunMacC}" == 1)
    Echo "C  {Cfile}  Exec Done">>"{Efile}"
End
If ("{RunLink}" == 1)
    Catenate "{Lfile}" >>"{Efile}"
    Delete -i "{Lfile}"
End
If ("{RunTMLLink}" == 1)
    Catenate "{Tfile}" >>"{Seqfile}"
    Delete -i "{Tfile}"
End
If ("{RunMacC}" == 1 || "{RunLink}" == 1)
    Echo "∂"{MakeWD}{Efile}∂";">>"{Seqfile}"
```

```
    Echo "∂"{MPW}Applications:Exec∂";"›››"{Seqfile}"
End
Open -t "{Seqfile}"
Find = "{Seqfile}"
Clear \;\ "{Seqfile}"
Echo "∂n∂}∂n∂);"›››"{Seqfile}"
Close -y "{Seqfile}"
Rez -o "{MPW}Applications:PasRunner" "{Seqfile}"
If {Status} == 0
    Delete -i "{Seqfile}"
End
Unset RunMacC
Unset RunLink
Unset RunTMLLink
Unset Lfile
Unset Cfile
Unset Tfile
Unset MakeWD
Unset Seqfile

### Set up the Resume tasks for after the applications
### Primarily for error handling
Set PostApp "{MPW}UserResume"
Echo "Unset Efile" >"{MPW}UserResume"
Catenate  "{MPW}MakeSupport:MixMakeResume" >>"{MPW}UserResume"

### If Diagnostic menu item is still around from
###  an earlier build cycle, delete it.
Set __oldExit {Exit}
Set Exit 0
DeleteMenu File "Open Diagnostics…" ≥Dev:Null
Set Exit {__oldExit}
Unset __oldExit

### Run applications using the custom Sequencer
"{MPW}Applications:PasRunner"
##########################################

###      File MixMake.Template.1
###  MixMake.Template.1
### Sample makefile for version 1 MixMake Command
###      W.G. Powell   1987 for MacTutor

### Make variable definitions
RezOpt = "-t 'RSRC'"

### Directory dependency rules go here
# This one specifies that rels are
# in sister directory to that of the sources
::Rels:  ƒ    :
# Can add others of form
# {AltSrcDir}:Rels: ƒ {AltSrcDir}

### Default dependency rules for compilers
# Default C compilation rule
.Rel ƒ    .c
    Delete -i {TargDir}{Default}.Rel
    Echo {DepDir}{Default}.c >> "{Cfile}"
    Set RunMacC 1
# Default assembly rule
.Rel ƒ    .Asm
    Delete -i {TargDir}{Default}.Rel
    echo {DepDir}{Default}.Asm >> "{Cfile}"
    Set RunMacC 1
# Default Pascal compilation rule
.Rel ƒ    .Pas
    Delete -i {TargDir}{Default}.Rel
    echo "Pascal  {DepDir}{Default}.Pas  Exec Exec"∂
        >> "{Efile}"
# Default resource compilation rule
.Rsrc    ƒ    .R
    Delete -i {TargDir}{Default}.Rsrc
    Rez {RezOpt} -o {TargDir}{Default}.Rsrc
{DepDir}{Default}.R

### Program-specific dependency rules go here
### For a program, should explicitly list the
### link script (.Link file), all objects (.Rel files),
###  and all resource files (.RSRC) needed to build.
# Sample dependency rule for Consulair linker
#  prog1  ƒ  prog1.Link prog1.Rel sub1.rel prog.Rsrc
#       Echo "Link prog1.Link  Exec  Done">>"{Lfile}"
#       Set RunLink 1
#
# Sample dependency rule for TML Linker
#   prog2  ƒ  prog2.Link  prog2.rel  lib.rel prog.RSRC
#       Echo "TML Link  prog2.Link  Exec  Done">>"{Lfile}"
#       Set RunTMLLink 1

##################################################

###          File MixMake.Template.2
###  MixMake.Template.2
### Sample makefile for version 2 MixMake Command
###      W.G. Powell   1987 for MacTutor

### Make variable definitions
RezOpt = "-t 'RSRC'"

### Directory dependency rules go here
# This one specifies that rels are
# in sister directory to that of the sources
::Rels:  ƒ    :
# Can add others of form
# {AltSrcDir}:Rels: ƒ {AltSrcDir}

### Default dependency rules for compilers
# Default C compilation rule
.Rel ƒ    .c
    Delete -i {TargDir}{Default}.rel
    Echo {DepDir}{Default}.c >> "{Cfile}"
    Set RunMacC 1
# Default assembly rule
.Rel ƒ    .Asm
    Delete -i {TargDir}{Default}.Rel
    echo {DepDir}{Default}.Asm >> "{Cfile}"
    Set RunMacC 1
# Default Pascal compilation rule
.Rel ƒ    .Pas
    Delete -i {TargDir}{Default}.Rel
    Echo "∂"{DepDir}{Default}.Pas∂">>"{Seqfile}"
    Echo "∂"{MPW}Applications:Pascal∂">>"{Seqfile}"
# Default resource compilation rule
.Rsrc    ƒ    .R
    Delete -i {TargDir}{Default}.Rsrc
    Rez {RezOpt} -o {TargDir}{Default}.Rsrc  ∂
                    {DepDir}{Default}.R

### Program-specific dependency rules go here
### For a program, should explicitly list the
### link script (.Link file), all objects (.Rel files),
###  and all resource files (.RSRC) needed to build.
# Sample dependency rule for Consulair linker
#  prog1  ƒ  prog1.Link prog1.Rel sub1.rel prog.Rsrc
#       Echo "Link prog1.Link  Exec  Done">>"{Lfile}"
#       Set RunLink 1
#
# Sample dependency rule for TML Linker
#  prog2  ƒ  prog2.Link  prog2.rel  lib.rel prog.RSRC
#    Echo "∂"{full directory}prog2.Link∂";">>"{Seqfile}"
#    Echo "∂"{MPW}Applications:TML Link∂";">>"{Seqfile}"
#    Set RunTMLLink 1

##############################################

###          File Done.c

##############################################
```

```
###          File Done.c
    /************************************************/
    /*                  Done.c                      */
    /*  Bogus Application for Exec to use to        */
    /*       Return to MPW shell                    */
    /*       Written for Consulair MacC             */
    /************************************************/
*Options E G H
main ()
{
void exit() ;
exit (0) ;
}
################################################

###      File MixMakeResume
# MakeResume - a command file to be executed when
#    resuming from applications such as MacC
#        W. PowellMay 1987 for MacTutor

# Don't terminate command file on non-zero status
Set __oldExit {Exit}
Set Exit 0
Set CompErrDir "`Directory`"
Export CompErrDir
# Establish a menu item for error handling
DeleteMenu File "Open Diagnostics..." ≥ Dev:Null
AddMenu File "Open Diagnostics.../E" 'Set Exit 0; ∂
Export MakeErr ; Set __origDir "`Directory`" ; ∂
Directory {CompErrDir} ;∂
CompErr ;∂
Set MakeErr {Status} ; Directory {__origDir} ;∂
Unset __origDir'
# {MakeErr} is a flag used by CompErr, the command
#    file which opens compiler error files
Set MakeErr 0
Export MakeErr
# Create summary files listing all files containing
#    current compiler or linker diagnostics
echo  '' › TMLP.Fer
echo  '' › Link.Fer
Files ≠.Pas ›Dev:Null ≥Dev:Null
If ( {Status} == 0 )
    For __fil In ≠.Pas ""
        If ("{__fil}" != "")
        Make {__fil} -f "{MPW}"MakeSupport:MakePErrs.mk
        End
    End › MakeOutApps ≥≥Dev:Null
End
Files ≠.Link ›Dev:Null ≥Dev:Null
If ( {Status} == 0 )
    For __fil In ≠.Link ""

        If ("{__fil}" != "")
            Make {__fil} -f "{MPW}"MakeSupport:MakePErrs.mk
        End
    End ›› MakeOutApps ≥≥Dev:Null
End
Unset __fil
Files MakeOutApps ›Dev:Null ≥Dev:Null
If ( {Status} == 0 )
    Execute MakeOutApps
    rm MakeOutApps
End
# Move insertion point to front of each
#    summary file window and save window
#    with that selection
Begin
    Open -t MACC.Fer
    Find o MACC.Fer
    Replace § ' ' MACC.Fer
    Save MACC.Fer
    Close MACC.Fer
    Open -t TMLP.Fer
    Find o TMLP.Fer
    Replace § ' ' TMLP.Fer
    Save TMLP.Fer
    Close TMLP.Fer
    Open -t Link.Fer
    Find o Link.Fer
    Replace § ' ' Link.Fer
    Save Link.Fer
    Close Link.Fer
End ≥Dev:Null
# Reset {Exit} variable to previous value
Set Exit {__oldExit}
Unset __oldExit
################################################

###          File MakePErrs.mk
# Make file to create error summary files from
# build cycles with TML Pascal and Consulair and TML Linkers
# This makefile is used by MixMake command's Resume tasks
#  (see file MixMakeResume)
#            W. Powell 1987 for MacTutor

#    These dependency rules ensure we get only
#    source files which
#    have NOT been changed since the error files
#    were written.
.Pas ƒ    .PErr    # for TML Pascal
    echo {DepDir}{Default}.Pas∂{DepDir}{Default}.PErr ∂
                ›› TMLP.Fer
.Link    ƒ    .LErr    # for Consulair and TML Link
    echo {DepDir}{Default}.Link∂    {DepDir}{Default}.LErr ∂
                    ››Link.Fer
################################################

###          File CompErr

### Shell script to find and open error files
### generated by MixMake
### command, opening a different files in sequence
### on subsequent
### calls.  ".Fer" files should have been created by MixMake's
### Resume tasks.  This script MUST be run by the menu item
### provided by the Resume task to work properly.
###    {Status} returned by this command on exit is the new
### value for global shell variable {MakeErr}.
###          W.G. Powell   1987 for MacTutor

# Some commands intentionally return nonzero status, so
#    don't terminate this command file
Set _oldExit {Exit}
Set Exit 0
###          Part 1
### Process MacC error files
Files MACC.Fer ›Dev:Null ≥Dev:Null  # Don't need output
If ({Status} == 0 && {MakeErr} == 0 )
# File MACC.Fer exists   AND still processing Mac C errors.
    Open -t MACC.Fer
    Find /'Errors in File: '/∆:∆/' ('/ MACC.Fer
        # Open bad source
    If ({Status} == 0)
        # find directory of source file
        Set __tmpSrcDir "`Catenate MACC.Fer.§`"
        Open -t -n MACC.tmp.tmp ≥Dev:Null
        Echo -n "{__tmpSrcDir}"›MACC.tmp.tmp ≥Dev:Null
        Find = MACC.tmp.tmp ≥Dev:Null
        Clear \∂:\∆:= MACC.tmp.tmp ≥Dev:Null
        Find o:= MACC.tmp.tmp ≥Dev:Null
    Set __tmpSrcDir "`Catenate MACC.tmp.tmp.§`" ≥Dev:Null
        Close -n MACC.tmp.tmp
        # open the file
        Open  "`Catenate MACC.Fer.§`"
        If ({Status} ≠ 0)
    Alert "Cannot open source file `Catenate MACC.Fer.§`"
```

```
        End
    Find /'(See '/∆:∆/')'/ MACC.Fer # Open diagnostic file
        If ((Status) == 0)
            Open "`Catenate MACC.Fer.§`" ≥Dev:Null
            If ((Status) ≠ 0)
                # try again in source directory
            Open "(__tmpSrcDir)`Catenate MACC.Fer.§`"
                If ((Status) ≠ 0)
        Alert "Cannot open error file `Catenate MACC.Fer.§`"
                End
                Unset __tmpSrcDir
            End
        Else
            Alert "Don't know corresponding error file"
        End
    Else # Open file MACC.Fer after processing all
            # the files it named
        Open MACC.Fer
        If ((Status) ≠ 0)
            Alert "Cannot open error file MACC.Fer"
        End
Alert "Finished last Consulair C Error File.∂nTML Pascal Next"
        # Save current selection
        Find §∆ MACC.Fer ≥Dev:Null
        Replace § " " MACC.Fer ≥Dev:Null
        Save MACC.Fer ≥Dev:Null
        Close -y MACC.Fer ≥Dev:Null
        Exit 'Evaluate (MakeErr) + 1'
    End
    # Save current selection
    Find §∆ MACC.Fer ≥Dev:Null
    Replace § " " MACC.Fer ≥Dev:Null
    Save MACC.Fer ≥Dev:Null
    Close -y MACC.Fer ≥Dev:Null
    Exit (MakeErr)
Else If ((MakeErr) == 0)
    Set MakeErr 'Evaluate (MakeErr) + 1'
    # Go on to Pascal files
End
###          Part 2
### Process TML Pascal Error files
Files TMLP.Fer ››Dev:Null ≥Dev:Null
If ((Status) == 0 && (MakeErr) == 1)
    Open -t TMLP.Fer
    Find /O/∆:∆/∂ / TMLP.Fer
    If ((Status) == 0)
    Open "`Catenate TMLP.Fer.§`"   # Open bad source
        If ((Status) != 0)
        Alert "Cannot open source file `Catenate TMLP.Fer`"
        End
        Find /∂  /∆:/▪/   TMLP.Fer
        If ((Status) == 0)
    Open "`Catenate TMLP.Fer.§`"   # Open diagnostic file
            If ((Status) != 0)
        Alert "Cannot open error file `Catenate TMLP.Fer.§`"
            End
        Else
            Alert "No corresponding error file"
        End
    Else
Alert "Finished last TML Pascal Error File.∂nLink diagns
next."
        # Save current selection
        Find §∆ TMLP.Fer ≥Dev:Null
        Replace § " " TMLP.Fer ≥Dev:Null
        Save TMLP.Fer ≥Dev:Null
        Close -y TMLP.Fer ≥Dev:Null
        Exit 'Evaluate (MakeErr) + 1 '
    End
    # Save current selection

    Find §∆ TMLP.Fer ≥Dev:Null
    Replace § " " TMLP.Fer ≥Dev:Null
    Save TMLP.Fer ≥Dev:Null
    Close -y TMLP.Fer ≥Dev:Null
    Exit (MakeErr)
Else If ((MakeErr) == 1)
    Set MakeErr 'Evaluate (MakeErr) + 1'
    # Go on to linker errors
End
###          Part 3
### Process Linker Error files
Files Link.Fer ››Dev:Null ≥Dev:Null
If ((Status) == 0 && (MakeErr) == 2)
    Open -t Link.Fer
    Find /O/∆:∆/∂ / Link.Fer
    If ((Status) == 0)
        Open "`Catenate Link.Fer.§`"   # Open bad source
        If ((Status) != 0)
        Alert "Cannot open link script `Catenate TMLP.Fer`"
        End
        Find /∂  /∆:/▪/   Link.Fer
        If ((Status) == 0)
        Open "`Catenate Link.Fer.§`"      # Open diagn files
            If ((Status) != 0)
        Alert "Cannot open error file `Catenate TMLP.Fer.§`"
            End
        End
    Else
        Alert "All error files have now been processed!"
        # Save current selection
        Find §∆ Link.Fer ≥Dev:Null
        Replace § " " Link.Fer ≥Dev:Null
        Save Link.Fer ≥Dev:Null
        Close -y Link.Fer ≥Dev:Null
        DeleteMenu File "Open Diagnostics…"
        Exit 'Evaluate (MakeErr) + 1 '
    End
    Exit (MakeErr)
Else If ((MakeErr) == 2)
    Alert "All error files have now been processed!"
    DeleteMenu File "Open Diagnostics…"
    Exit 'Evaluate (MakeErr) + 1'
End
Exit (MakeErr)
################################################

###          File QuitPermVars
#### This shell script is run by the Quit script
#### to save any shell variables which must be
#### preserved through a Quit
#### W.G. Powell 1987 for MacTutor

            #Clear the file
            Echo "" › "(MPW)StartPermVars"

        # Save CompErr (Diagnostic viewer) state variables
Echo "# State vars for CompErr" ››"(MPW)StartPermVars"
            Set MakeErr ›› "(MPW)StartPermVars"
        Echo "Export MakeErr" ›› "(MPW)StartPermVars"
            Set CompErrDir ››"(MPW)StartPermVars"
    Echo "Export CompErrDir" ›› "(MPW)StartPermVars"
            If ((MakeErr) < 3)
                # Still some compiler errors to process?
                AddMenu File "Open Diagnostics…" ∂
                    ›› "(MPW)StartPermVars"
            End

################################################

###          File StartPermVars
#initial default values for CompErr state variables
Set MakeErr 3
Export MakeErr
Set CompErrDir ""
Export CompErrDir
################################################
```

```
###          File Man
###          MPW command "man" for on-line help
###              Written 1987 W.G. Powell
###                   for MacTutor
###      Search more than one file sequentially for help text

Set Exit 0
Set Reval 0
### Get standard command names from alias dictionary
# need temporary file
Set _dmfl_ dumrxfx_
#   Unalias the input arguments
Echo {Parameters} | d
  Canon "{MPW}bilITools:Alias.Dict" > "{_dmfl_}"
#   Set new input arguments
Set HelpList "`Catenate {_dmfl_}`"
Delete -n {_dmfl_}
###      Look for each requested item individually
For HelpItem In {HelpList}
    # Look in default help file first
    Help {HelpItem} ≥ Dev:Null
    Set Retval {Status}
    If ({Retval} == 2 || {Retval} == 1)
        # Look in local help file if not in default
        Help -f "{MPW}Local.Help" {HelpItem} ≥ Dev:Null
        Set Retval {Status}
    End
    If ({Retval} != 0)
        # Keep a list of items NOT found for diagnostic
        Set Reval {Retval}
        Set NotFnd "{NotFnd} {HelpItem}"
    End
End
###      Send error message if items not found.
If ({Reval} != 0)
    Echo "### Help:  Cannot find items:">Dev:StdErr
    Echo "###     {NotFnd}">Dev:StdErr
End
Unset NotFnd
Unset _dmfl_
Unset HelpItem
Unset HelpList
Unset Retval
Exit {Reval}
################################################

###              File Local.Help
Help files for locally developed commands
W.G. Powell 1987 for MacTutor

-
Local - MPW Local Custom Commands
    Get info on specific command by entering
    Man CommandName

CompErr      # Open compiler diagnostics and erroneous
             #sources
MacC     # Consulair applications: compiler, linker, etc.
MakeCPath# Use shell variables to create Consulair paths
MakePPath# Use shell variables to make TML paths
Man      # "help" utility including local help files
MixMake  # MPW make with Mac C, TML systems
NullFile     # A "do nothing" command file
TML      # TML Pascal compiler and linker
UpdatePaths  # Update TML Pascal, Mac C search paths

Man [{CommandName | Keyword) … ]
    Abbreviated on-line manual entries for commands.  This
    is an elaborate version of MPW help, which searches
    several files for help info.
-
UpdatePaths
    Create new search paths for Consulair and TML, but only
1) If {MPW}Startup (which contains path variables) changes
2) If {MacCLib}C.Path.Template (Path Manager input) changes
    SEE ALSO:  MakePPath, MakeCPath
-
MakePPath
    Create new "Paths File" for TML Pascal in system folder.
    Pathnames come from shell variables defined in Startup:
    {TMLPasLib} contains one directory for libraries
    {TMLPasInc} contains up to 4 directory  pathnames
        separated by commas.
    User must select Set Paths DA
    and click "OK" afterwards.
-

MakeCPath
    Create new "Paths.Rsrc" for Consulair in system folder.
    Also writes equivalent Path Manager text input file in
    system folder as "MacC.Path".  Input text file uses
    {MacCLib}C.Path.Template as main form, with additional
    pathnames from shell variables defined in Startup file:
    {MacCLib} contains one directory pathname for libraries.
    C.Path.Template names several paths relative to
    {MacCLib}.
-
MixMake [option…] target…
    Build programs using MacC and TML compilers
    Options:
      -d name[=value] # define variable name as value
                      #   (override definitions in makefile)
      -e              # rebuild everything regardless of dates
      -f makefile # read dependencies from makefile -
                      #   (default "MakeFile")
    Other standard Make options are not supported.
    The following options
    are helpful using the standard Make tool with the
    MixMake input dependency file
    -r   # write roots of dependency graph to output
    -s   # write structure of target dependencies to output
    -t   # touch dates of targets and prerequisities
    -u   # identify targets in makefile not reached in build
    -v   # write verbose explanations to diagnostics
    SEE ALSO: Make, CompErr
-
CompErr
(ALWAYS access via item "Open Diagnostics…" in File menu)
    Opens MacC, TML Pascal, or Linker diagnostic
    and source files.  Generally only will work following a
build
    attempt using command "MixMake".
    Opens only one pair of files on each invocation
    - must be called repeatly for each source.
-
NullFile
    Contains no command text.  Does nothing.
-
########################################
             File  Alias.Dict
File Target
```

-

News From the Romper Room Refugees

(a.k.a. Macintosh Developer Tech Support)

Chris Derossi
Fred Huxham
MacTutor Vol. 4 No. 3

Ask Apple Technical Support
by Max AppleZone

Welcome to the Macintosh Developer Tech Support column. We at Mac DTS will use this column each month to answer your questions, report on the events at Apple, and pass along new Mac programming information. This is your chance to raise issues and questions just because you're interested in programming the Mac; you don't have to be a certified developer working on the next Mac database.

The Mac Developer Tech Support group at Apple is a part of Developer Services, the organization whose job it is to promote and support third part developers. Tech Support's job includes writing TechNotes, creating sample code, generally helping to improve the quality of your code, and answering programming questions via electronic mail (either MCI Mail or our own AppleLink). This column is part of an effort to make our services more visible and available to the right people (you).

If you have a programming question or a topic that you'd like to see addressed in this column, send a note (letter, postcard, letterbomb) to MacTutor Magazine, at PO Box 400, Placentia, CA 92670, c/o Romper Room. We'll do our best to publish and answer your question.

If you are developing (or thinking about developing) a full-blown Mac application (commercial or otherwise), then you should think about joining our Certified Developer Program. Certified developers get put on our mailing list and have access to MacDTS to help answer their questions on a personal level. To inquire about becoming a Mac Certified Developer, contact the Developer Programs group at Apple Computer at (408) 973-4897.

Since we don't have your questions yet, this month's column contains information that was presented at the January MacWorld Expo in San Francisco. The first is Chris Derossi's presentation on MultiFinder compatibility, and the second is from the handout from Fred Huxham's presentation on writing code to be compatible with Apple's AU/X, both given at the MacTutor session at the San Francisco MacWorld Expo last January.

MultiFinder Programming Tips by Chris Derossi
Be MultiFinder Aware

Generally, being compatible with MultiFinder is easy, and mainly involves following the guidelines outlined in *Inside Macintosh* and the TechNotes. Being MultiFinder aware is another issue. MultiFinder awareness is actually not that difficult, and requires just a few things: a SIZE resource that describes your application's knowledge of MultiFinder (see below), code to handle Suspend/Resume events, and sharing time with other applications by using WaitNextEvent.

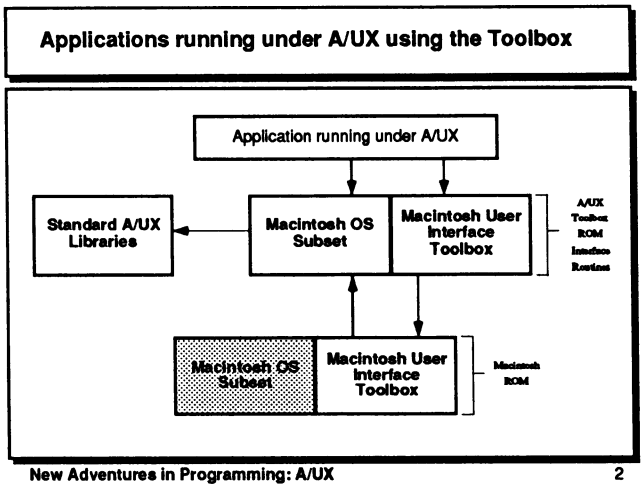


Fig. 1 Apple's A/UX environment

With Suspend/Resume events, your application can determine when it is running in the foreground or the background. In the background, your application should run as normal, but it shouldn't expect any events other than update events, and it shouldn't do anything that would pull the rug out from under the foreground application (like changing its cursor).

To do background processing, your application should divide its background tasks into small chunks. Each time that your application gets background time, it should do one piece of its task, and then relinquish the processor again by calling GetNextEvent or WaitNextEvent.

When your application is in the foreground and it is waiting for a user event, it should call WaitNextEvent. This gives processor time to the background applications.

Handle ALL Events

Under MultiFinder, your application cannot prevent itself from being switched out by masking Suspend/Resume events (App4Events). If you mask out Suspend events, you will still be switched out; you just won't know that it happened.

Be prepared to handle Update events at all times. While suspended, your windows may need updating when other windows are moved. MultiFinder will pass Update events to your suspended application until the update is performed (or until MultiFinder has tried so many times that it decides you're never going to handle the event).

Share The File System

Now that the Finder and other applications use the File System during the execution of your program, you have to pay particular attention to your use of files. The best tip is to always use the file's refNum, and never rely on the file name or the pathname of the file after it has been opened. Other applications,

© The Definitive MacTutor, Vol. 4

or even the user with the Finder, may rename and move the file after you have opened it, but the refNum will stay valid.

If you need to get the name or directory of an open file, call GetFCBInfo (not available on 64K ROMs) each time that you need it.

Since files that are not open can be deleted or changed, if you need a file to remain undeleted and unmodified, keep it open. If you open a file, read from it, then close it, the next time that you try to access that file, it may have been deleted, renamed, moved, modified, or opened by someone else.

And above all, CHECK ERRORS! Because you're sharing the File System, you may not have exclusive use of it. This means that, even though you have only a few files open, for example, you may get a tmfoErr (Too Many Files Open Error). Similarly, you could get a tmwdoErr (Too Many Working Directories Open Error). And even if you've closed all of your files on a volume, UnmountVol will now return an error if other files are open.

Use The Right A5

In general, register A5, which points to your QuickDraw world and to your application's global variables, is only valid when your application's main code is running. Auxiliary pieces of code that are separate from the application (like MDEFs, VBL tasks, Time Manager routines, completion routines, etc) are not guaranteed to have your application's A5; they may be executed while A5 points to some other application's globals.

When such code gets executed, you should first save register A5, and then load A5 with the one from your application. This will let you get to your application's variables, routines, and QuickDraw world. When your code has completed, be sure to restore A5 to the value it had when your routine was called.

Use The Clipboard For Cut, Copy, and Paste

Since many applications may be running, your application should not use the scrap for private data storage. If your application changes the scrap on its own, not in response to a user's Cut or Copy, the user may see the Clipboard change unexpectedly in other applications.

On the other hand, by fully supporting Cut, Copy, and Paste with the scrap, your users will be able to easily move data among applications.

Understand When & What Happens WhenSwitching

When MultiFinder does a context switch from one application to another, it swaps out the following things:

- Low-Memory—System-Wide Globals
- 68000 Registers (Including A5 and A7 for globals and the stack)
- The Application's Layer—All of the application's windows
- Trap Patches that were made from within the application
- VBL Tasks in the application heap
- Drivers in the application heap
- Time Manager tasks in the application heap

There are three cases where MultiFinder will do a context switch. The first, and most obvious case occurs when the user clicks on a suspended layer, chooses a suspended application from the Apple menu, or clicks on the small icon in the right side of the menu bar. This causes a full context switch, bringing the resuming application's layer to the front.

This full context switch won't occur under two conditions: when the front window of the foreground application is a dBoxProc (standard modal dialog window), and when there are pending file system calls. (When the file system calls complete, the switch will happen.)

MultiFinder checks the front window's proc ID to determine if it's a modal dialog. So dBoxProc windows should only be used for modal dialogs. Using this window type for other purposes will block the switching mechanism. (If you like the look of the dBoxProc, you can use a plainDBox, and draw the additional thick line yourself.)

The second two context switches can happen ANY time that GetNextEvent, WaitNextEvent, or EventAvail is called, even from a modal dialog. Neither of these switches changes the ordering of the layers, or generates Suspend/Resume events. One switch is for giving time to background applications, and the other is for passing update events to suspended applications. MultiFinder will only give background time to those applications that have the CanBackground bit set in their SIZE resource. ALL applications get background update events as needed.

The SIZE Resource

Even if your application is not going to do background processing, it should have a SIZE resource. In addition to telling MultiFinder what MF features your application supports, the SIZE resource also specifies how much memory your application needs. If you don't have a SIZE resource, MultiFinder gives it a default partition size of 384K, which may be way too much or way too little for your app.

The SIZE resource must have an ID of -1 and contain the following data:

Flags	Integer (16 bits)
Preferred size	LongInt
Minimum size	LongInt

The flags word is a series of 5 boolean values and 11 unused bits. All unused bits MUST be set to zero for future compatibility. The bits are assigned as follows:

Bit 15 **dontSaveScreen/saveScreen**

This is included for Switcher compatibility only

Bit 14 **ignoreSuspendResumeEvents/acceptSuspendResumeEvents**

Set this bit if your application understands Suspend and Resume events.

Bit 13 **enableOptionSwitch/disableOptionSwitch**

This is included for Switcher compatibility only

Bit 12 **cannotBackGround/canBackGround**

Without setting canBackground, your application will not get nullEvents while it is suspended.

Bit 11 **dontDoOwnActivate/doOwnActivate**

If this bit is set, then your application accepts the responsibility for deactivating and activating its own windows when the application is suspended and resumed. This field may also be called notMultiFinderAware/MultiFinderAware. This bit should only be set if bit 14 is also set.

The preferred and minimum sizes are the number of bytes that your application needs. So if your application runs best with 512K of RAM, but will survive with only 128K of RAM, then the

preferred and minimum sizes would be 524288 (512K) and 131072 (128K).

How to be A/UX Friendly
by Fred Huxham

In figure 1 we see how applications running under Apple's version of Unix, called A/UX, can use the Macintosh toolbox to implement the Macintosh User Interface Guidelines. The Unix program makes a toolbox call, which is routed to a A/UX toolbox ROM interface routine, which in turn calls the trap in the Macintosh ROM. A subset of the Macintosh OS traps are also provided with an interface, so that OS traps get routed to standard A/UX libraries, to perform the expected Unix function, both for OS calls from the toolbox, and from the User's application. The purpose of the Mac interface is to provide an environment that will run the following:

- Macintosh Binaries
- Unix Applications with toolbox enhancements
- Macintosh / A/UX Applications

Here are some things to keep in mind when designing for A/UX compatibility:

- Differences in execution environments
- Memory Manager
- File Manager
- Differences in C Compilers
- Differences in Language Conventions

Here are some more notes on the remaining slides.

Slide #5

• 32-bit Address Violations

Directly accessing the flag bits on relocatable blocks is a fate worse than death. The Macintosh OS only uses the low-order 24 bits of an address. The high-order 8 bits are used as flag bits (Lock, Purge, Resource). A/UX however, uses full 32-bit virtual addressing. If an application directly manipulates one of the upper 8 bits of an address under A/UX, it invalidates the address.

• CPU Privileged Instructions

The A/UX Toolbox is run by an A/UX process in User Mode. Because of this, applications cannot make privileged instructions. Some MacOS applications move to and from the status register. If the motivation is to access the condition code bits, it is possible to do this by using the MOVE CCR,<ea> instruction. Because this is only available on a 68020, you must check to see that is what you are running on. If you are accessing the SR to turn interrupts on or off, you're out of luck, only the kernel can do that.

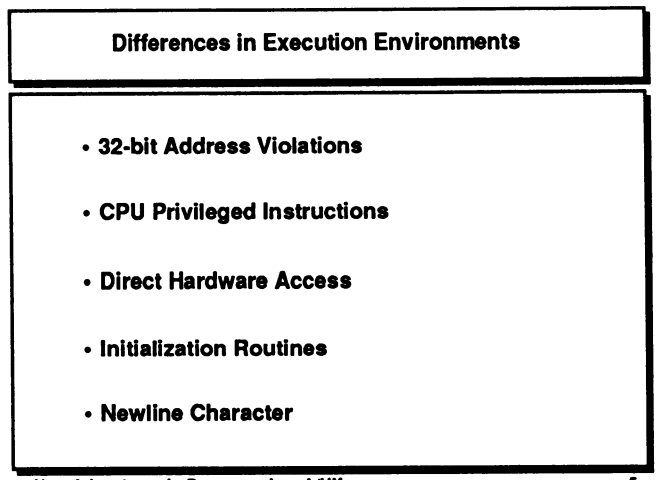
• Direct Hardware Access

Under A/UX only the kernel is allowed direct access to the hardware. This means applications cannot:

- Access the serial port by talking to the SCC's registers.
- Access the disk drive by talking to the disk controller chip.
- Access any of the low-memory CPU exception vectors.

In addition, not all the low-memory globals are valid under A/UX. Most that are not related to hardware are supported.

• Initialization Routines



Make sure call InitDialogs and TEInit.

•Newline Character

In the Macintosh environment, lines are terminated with a return (0x0D). In the A/UX environment, lines are terminated with a linefeed (0x0A).

Slide # 6

• Heap Zones

There is no distinction between the system heap zone and the application heap zone. The routines SystemZone and ApplicZone return the same pointer.

• Master Pointers

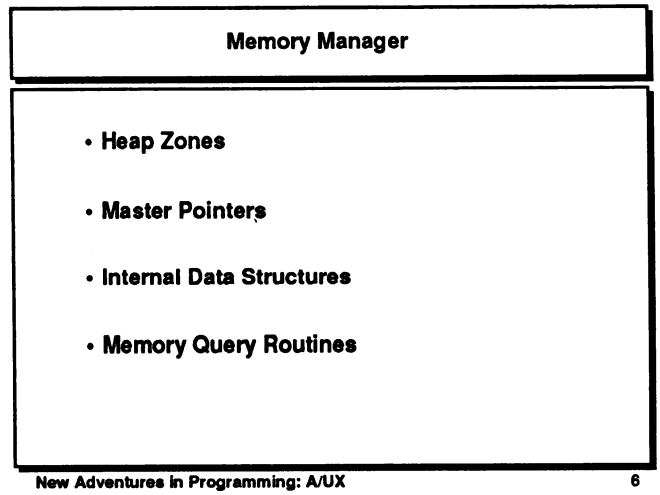
Under the A/UX OS, master pointers are 8 bytes long, rather than the 4 bytes used by the Macintosh OS.

• Internal Data Structures

The internal data structures of the Memory Manager are different from those in the MacOS. The heap zone header record is the same, but not all fields are used. An application should not attempt to traverse the blocks in a heap zone.

• Memory Query Routines

A/UX supports virtual memory, but the Mac memory manager routines are designed for a system that has a known amount of physical memory. At the current time, memory query routines such as FreeMem return the amount of memory that would be



available if your application had started with 1 MB of free memory.

Slide # 7

• Filename Conventions

Applications running under A/UX must follow the A/UX filename conventions:

- 14-character names.
- The character / instead of : to separate directories.
- A smaller range of legal characters.
- Case sensitive filenames.

• Resource and Data Files

In the MacOS, both resources and data appear in separate forks of the same file. In A/UX, resources appear in a separate file, with the filename extension ".res".

• Volumes

Under A/UX, all Macintosh files exist on a single, emulated Mac volume.

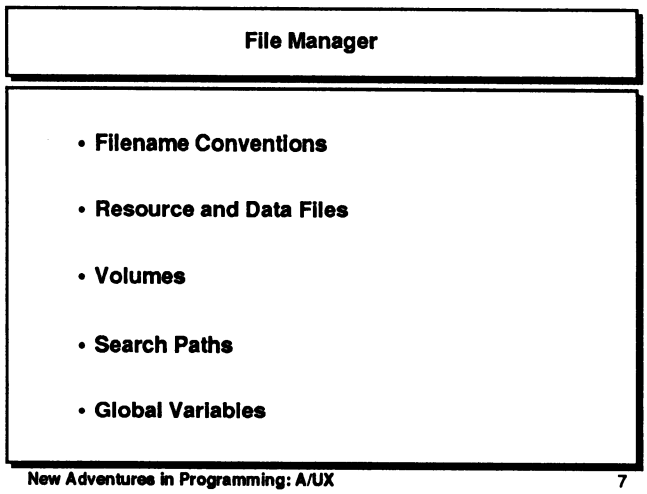
• Search Paths

Unlike the MacOS, under A/UX a file must be in the specified directory or it will not be found. No alternative paths are searched.

•Global Variables

Only the following File Manager low-memory globals are supported:

BootDrive
VCBQHdr
DefVCBPtr
DrvQHdr
FSFCBLen



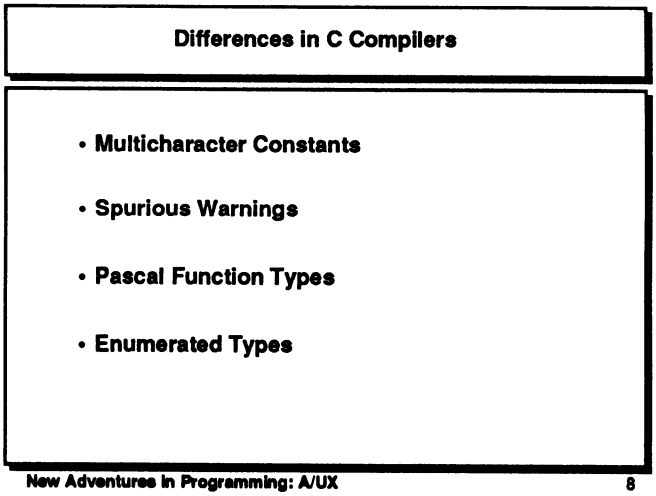
Slide # 8

• Multicharacter Constants

The two C compilers evaluate character constants in opposite order. For example, the A/UX C compiler converts 'ABCD' to 0x44434241, and the MPW C compiler converts it to 0x41424344. Multicharacter constants are used commonly to define resource types.

• Spurious Warnings

The A/UX compiler generates warnings when it sees zero-length arrays, which appear frequently in the include files.



• Pascal Function Types

MPW C has an extern pascal function type used for calling most of the ROM routines. To use these same functions, an A/UX C program must use assembly glue.

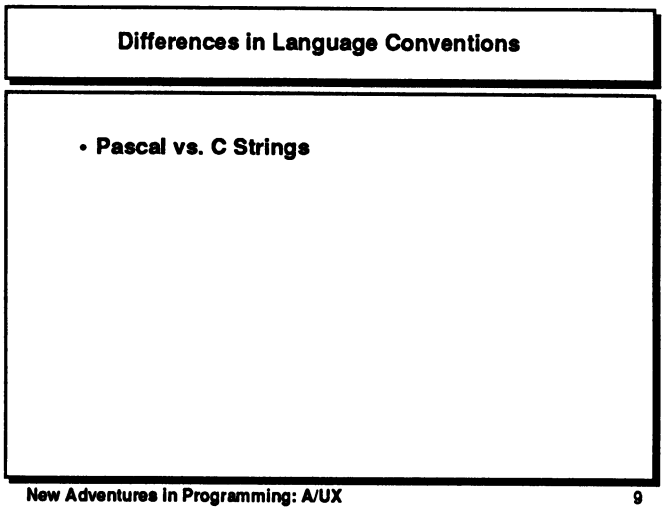
• Enumerated Types

In MPW C, enumerated types can be 8, 16, or 32 bits long, depending on the range of possible values. In A/UX C, enumerated types are 32 bits long (unless packed in structures using bitfields).

Slide # 9

• Pascal vs. C Strings

Many of the ROM routines follow Pascal conventions for storing strings, pushing parameters on the stack, and returning function results. the conventions differ from standard C conventions. The A/UX Toolbox includes conversion code that takes care of most of these incompatibilities as long as you use A/UX Toolbox routines for storing and retrieving data. However, strings in resource files and in A/UX Toolbox data structures are all stored as Pascal strings.



VIP Views
Calculating Distance for Navigation

TextEditor

Bill Luckie
Contributing Editor
Simi Valley, CA
MacTutor Vol. 4 No. 3

V.I.P. 2.2

Mainstay has released version 2.2 which is a significant upgrade and is free to all registered owners. V.I.P. 2.2 includes 25 additional procedures, improved keyboard control, 5 new intrinsic functions to provide added flexibility in handling Point and Rectangle objects, and the ability to create a true "About..." under the Apple menu. If you are a V.I.P. owner and haven't sent in your registration card, you are missing some neat features.

In the beginning

In a previous article (April 1987), I presented a 'shell' program which implemented some of the common user interface actions e.g., Menus, Windows, File handling, and Events. In this program you will see that the same routines are used. In some cases these routines have been expanded to handle the specifics required of our program, and of course additional routines have been added. The only change brought about by changes to V.I.P. itself is the 'About' routine, which is now called Gee_Whiz. As mentioned earlier V.I.P. programmers can now create a "About (program name)" entry under the Apple menu and have it display whatever they choose. In this case selecting 'About GC_Dist...' displays the same window and information as presented at program startup.

The guts and feathers

The purpose of this program is to illustrate by example some of the most commonly used procedures. In addition to the already explored subjects we will create Dialogs, obtain user input data organized as records and save them to disk, work with some of V.I.P.'s intrinsic functions, and most importantly, have some fun programming. As is the case with most programs there is certainly more than one way to accomplish a particular objective. What I present here is not advocated as the only way, or even the best way, to do the job. In fact I violate a few of the rules set forth in *Inside Macintosh*, and I'll tell you about them as we proceed.

How far is it?

This program computes and displays the Great Circle distance between any two cities in the data file. Figure 4 contains the data required, which is input using the dialog as shown in figure 1. The Great Circle distance, in case you didn't know, is the shortest distance between any two points on a sphere. It is

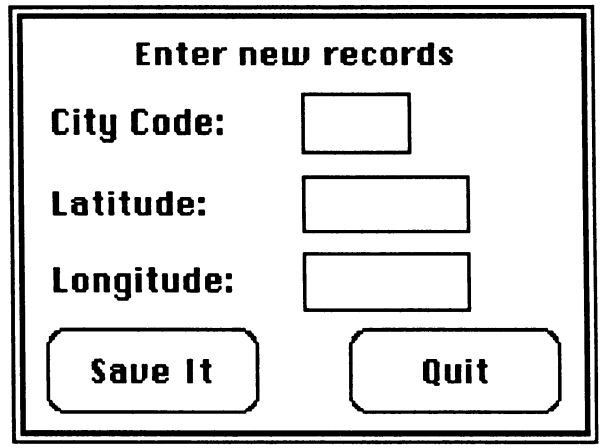


Figure 1 Our Dialog Box

most commonly used to represent the distance in nautical miles between airports for payload/range, performance, and fare calculations. Airports are each identified by a unique three character code. Los Angeles International airport, for example is LAX. The position on the globe of each airport is given as a latitude and longitude usually expressed in degrees and minutes. Latitudes south of the equator, and longitudes west of Greenwich, England are identified as negative angles. To work with these data, degrees and minutes are converted to decimal degrees, and to cope with SANE's handling of some intrinsic functions, a conversion of decimal degrees to radians and vice is also required.

Learning to love rectangles

Three dialogs are used in this program. Record_Dialog, which is used to input the city code, latitude and longitude data. Edit_Dialog, which allows viewing and editing of the records previously saved to disk, and Input_Dialog for the user to specify the origin and destination city codes as input to routines for calculating the great circle distance.

V.I.P. allows the programmer to create dialogs using built in procedures as I have done in this example, or as resources created with a resource editor such as ResEdit or Dialog Creator. The difficulty in using V.I.P.'s procedures to create dialogs is in the visualization and placement of the various rectangles which bound each dialog item. Dialogs can contain static text, editable text, radio buttons, check boxes, and ordinary buttons, each situated within a rectangle. Creating complicated dialogs using V.I.P.'s procedures is enough to discourage ordinary mortals, but there is an easy solution even if you don't care about resources.

My approach is to use Dialog Creator. Dialog Creator by

© The Definitive MacTutor, Vol. 4

Michael Bayer of Apple Canada lets you create dialog templates and dialog item lists interactively. It's a simple matter to draw the dialog then print the text source file which gives you the coordinates of each rectangle and the assurance that your dialog will look OK.

For each of the three dialogs used in this program the code is essentially the same. A rectangle is specified for the size and location of the dialog window, and the *new dialog* procedure follows with arguments for rect, title, and dialog name. Entering a text string as the title argument indicates the dialog is to be non modal. *Set rect* and *append dialog item* procedures are now repeated for each item in the dialog. When referring to dialog items, their order of appearance in the dialog item list (DITL) is used.

By convention, dialog item #1 is the default action to be performed when the user presses the return or enter key generally signifying OK. (Note; this default action is implemented automatically in V.I.P. for modal dialogs only) When all dialog items have been appended to the list, the *open dialog* procedure displays the dialog on the screen. If you prefer to create dialogs as resources, then you will need to change all occurrences of *kill dialog* to *close dialog* so as to not erase them from memory until Quit is selected.

DoDialogEvent

Now that we have our dialogs created, we need to be able to process the information entered or displayed according to the users action. For example, in the dialog as shown in figure 1, you can see there are three editable text fields and two buttons. Data entered by the user in the text edit fields is saved to disk when the 'Save it' button is clicked, or conversely, the dialog is dismissed whenever the 'Quit' button is clicked.

The DoDialogEvent routine receives two input arguments, i.e. Dialog and item to direct the action to be taken depending on which dialog is active and which item of the dialog was selected. In this case the first *if* logic form is true, and program flow is directed to the Get_Record routine when item=1, and to the *kill dialog* procedure whenever item=2. Actions to be performed by the other dialogs are similarly handled.

The purpose of handling all dialog events in one routine is probably not apparent in this small program. However, this routine like those presented in my previous article, are really generic, in that they don't do any real work, but merely direct program flow to a routine specifically designed to perform a discrete task. Programs written using a structured approach are usually easier to modify and are certainly easier to debug.

Get_Record

This routine when called by the DoDialogEvent routine simply executes three *get dialog item* procedures corresponding to the three text edit fields in the Record_Dialog. If you count the items appended to Record_Dialog you will see that items 7, 8, and 9 are the text edit fields receiving user entries for City Code,

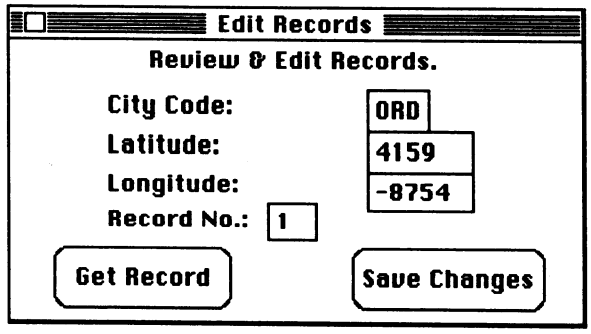


Figure 2 Editing our distance data base

Latitude, and Longitude respectively. To assure the City Code is stored properly, the object 'City' is converted to upper case. When this routine has completed its miniscule tasks, program flow resumes in the DoDialogEvent routine and the Store_Records routine is called.

Store_Records

With the information the user has entered in Record_Dialog extracted by the Get_Record routine as a result of the user clicking the 'Save It' button, we are now going to save it to disk as a record. A record, as I'm sure you know, is a group of related data organized in fields. In V.I.P., records themselves are identified by a Record Number, and individual fields are located within a record by specifying the offset (the number of bytes) from the beginning of the record. In preparation for storing the record to disk, the program decides if this is the first time its been asked to store a record, and if so, to display the standard file dialog, otherwise just save the record to disk without further ado. The first procedure sets the value of n according to how many times the routine is accessed, and is used as an argument to the object RecNo[n] which serves to automatically number our records.

The *if* (flag) logic form checks for true and directs execution flow accordingly. So, let's say this is the first record to be stored. In that case flag=0 or false, and execution continues with the first *else* logic form. The *get document name* procedure displays the standard file dialog with the prompt, "Save file as 'Cities'". Here's my first transgression from the scriptures of *Inside Macintosh*. Rather than allowing the user to name the file anything he (she) wishes, I tell 'em name it 'Cities'. If this offends your sensibilities, please feel free to change the code. Actually, you may name the file anything you choose. The argument *name* serves equally as the input argument, and its contents appear as the default name in the edit field reserved for the name of the file to be saved. To change the default name, initialize the global object 'FileName' to whatever you prefer, and don't' forget to set the array size at least equal to the string length plus 1.

The user accepts 'Cities' as the name of the file to be saved and clicks OK to dismiss the dialog, and the *open file* procedure

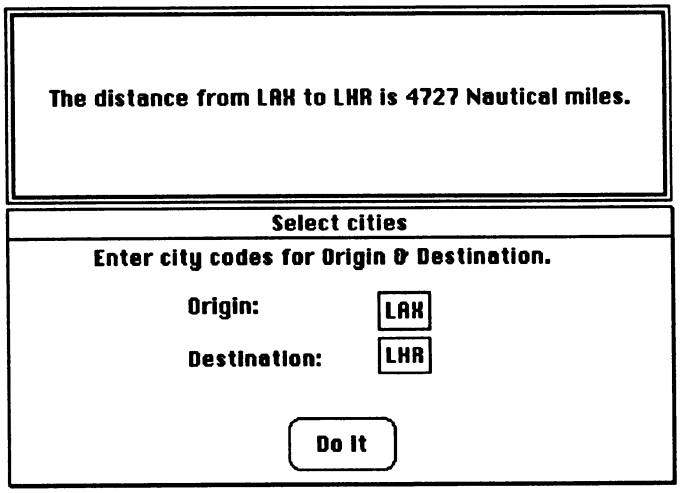


Figure 4 Output of our distance program

opens the file in the 'write' mode. The *allocate record* procedure allocates a record in memory with a length of 18 bytes, and identifies it with RecNo[n], which is now 1.

City code is always 3 bytes plus 1 byte for end of string marker, the maximum length of Latitude and longitude is 6 bytes plus an additional byte for end of string. All data making up the record is stored as a continuous text string with the record length being the sum of each field plus an additional byte.

Three *put field* procedures take the global objects, City, Lat, and Long, and position them within the record. Arguments for *put field* are; record number, byte offset, and field length. Byte offset is expressed in bytes and the first position in a record is byte position zero. Field length is also expressed in bytes and is sized to handle the longest entry anticipated.

Procedures *write record*, *free record*, and *close file* complete the task of saving the record to disk. Now, because this is the first time this routine has been called, we set the value of flag to 1 or true. An alert is displayed informing the user that additional records added during this session will be saved without the GetFile dialog. Three *set dialog item* procedures clear the previously entered data, and position the text edit caret in the first text edit field ready for entry of the next record.

Assuming the user continues by entering data for the next record; when the Save It button is clicked, program flow progresses exactly as before. The difference is, when Store_Records is called this time, the counter n is incremented and the first *if* logic form now evaluates to true. The *open file* procedure opens the file 'FileName', and the *get document length* procedure returns the length in bytes of the file in the object 'Position'. To avoid overwriting previously saved records, the *file position* procedure sets the file position according to the value of Position. From here on the same procedures are executed as before to save the record to disk.

Review_Records

Let's say, for purposes of this tutorial, that you have entered all of the data and you wish to see if it is correct. By selecting View or Edit Records... from the Options menu, a non-modal dialog is displayed as shown in figure 2, and the Review_Records routine is called. This short routine opens the file 'FileName' in the read only mode. A *for* logic form provides a loop to be executed 10 times (NoRecs is a constant assigned the value of 10). The *allocate record* and *read record* procedures contained within the *for* loop are thusly executed 10 times to bring the entire file into memory. For safety sake the file is closed, and the value of the object n is set to zero.

When the Get Record button is clicked, the DoDialogEvent routine calls the Next_Record routine. As you might have guessed, when the Save Changes button is clicked, DoDialogEvent processes the request by calling the Save_Changes routine. The dialog is ultimately dismissed by clicking the window's close box.

Next_Record

This routine is simply a series of *get field* and *set dialog item* procedures to display the contents of the records saved in the file 'FileName'. To start at the beginning of the file, the object n is assigned the value of 1. Record Number is displayed by converting the object n to a byte type array named string with the procedure *number to string*. Note, the minimum dimension for a byte type array to handle number to string conversions is 13.

The Next_Record routine is called each time the user clicks the Get Record button. When the last record has been viewed, the *assign* procedure within the *if* logic form is executed to set the value of the object n to zero. This merely allows the continuous viewing of records by starting again at record number 1 when the last record has been displayed.

CITY	CITY CODE	LATITUDE	LONGITUDE
Chicago	ORD	4159	-8754
Geneva	GVA	4614	607
New York	JFK	4038	-7347
San Francisco	SFO	3737	-12223
Los Angeles	LAX	3356	-11824
London	LHR	5128	-27
Cairo	CAI	3008	3124
Venice	VCE	4526	1221
Athens	ATH	3754	2344
Toronto	YYZ	4341	-7938

Figure 5 Our data base for distance calculation

Save_Changes

While viewing the records, changes may be made by entering the appropriate data in one or more of the text edit fields and clicking the Save Changes button. To no one's surprise, the Save_Changes routine is called by the DoDialogEvent routine whenever the Save Changes button is clicked.

This routine simply opens the file 'FileName' in the write mode and sets the file position to overwrite the record to be

changed. To do this, the procedure *file position* uses the argument (n-1)*18. Or in plain English, subtract 1 from the record number because the record pointer is left at the end of the record we wish to change. Now multiply by the record length (18), to set the file pointer to the correct position as calculated from the beginning of the file.

The actual writing of the record uses the same procedures as described previously. Two *if* logic forms are used to maintain correspondence with the actual record number, and *if* n=0 as is the case when the last record is the one to be edited, we simply assign the value of 10 to n. To make this a temporary measure, the object 'flag' is set to true, which is detected by the next if, and the value of n is once more zero. This only makes sense if you remember 'n' will always be something between 0 and 9 whenever this routine is called. So if n is equal to anything from 1 to 9, just write the record, else if n=0 change it to 10 temporarily. There must be an easier way.

Find_City

By selecting Compute Distance... from the Options menu, the dialog as shown in figure 4 is displayed. When the user enters the city codes for the origin and destination cities and clicks the Do It button, the Find_City routine is called by the Get_Input routine, and receives Origin and Destination as input arguments

The first thing this routine does is check to see if you are trying to fool it by entering the same city code for both the origin and destination cities. Normally an input like this should result in a distance of zero, but for some reason Apple's SANE doesn't like zero's as operands for certain intrinsic functions. So, rather than having the Find_City routine complete all of its tasks, and then trap the error in the Calculate routine, I simply chose to display an alert informing the user of the problem.

Find_City opens the file 'FileName', reads the records into memory, and then compares the City field in the first record with the input argument Origin. If match=0 (are the same), two *get field* and *string to number* procedures extract and convert the data stored as Latitude and Longitude to numbers for the Origin City. When the origin city has been found, flag1 is set to true, and program execution continues looking for a match with the Destination city.

In this manner, the maximum number of times the records are searched for both the Origin and Destination cities is equal to the number of records in the file. If either flag1 or flag2 is not set to true, an alert informs the user that his choice is not in the file, and execution of this routine is terminated with a return procedure.

When both Origin and Destination cities have been found, Latitude and Longitude of the Origin and Destination cities are now passed to the Calculate routine as A,B,C,D respectively, as are the Origin and Destination city codes.

Calculate

If you haven't fallen asleep by now, I'm sure you recall the data we stored as Latitude and Longitude represented degrees and minutes which were actually stored as byte arrays and later converted to real numbers by the Find_City routine. So, for example, the Latitude of LAX we entered was 3356 with the rightmost two characters representing the minutes portion in every case.

When the object A is received in this routine, and assuming LAX is the Origin city, it would appear as 3356.000 if we were to examine its value. To extract the minutes portion of this number we simply divide by 100 to move the decimal point two places to the left, and use the *int & fract* procedure to separate the integer and fractional portions so we can convert the fractional portion to decimal degrees. By dividing the fraction by 60 and multiplying by 100, and adding the fractional part now in decimal degrees to the integer we have a number we can almost work with.

This same series of manipulations is completed for objects B,C, and D, and now because SANE utilizes radians when working with sin, cos, and acos, we need to do another conversion. By multiplying each object A,B,C,D by (pi/180) we are finally ready to do the calculation to see how far it is between our Origin and Destination cities, however the answer wont make any sense until we convert the result 'Dist' to Nautical Miles. The distance is rounded off to the nearest whole number, converted to a string, and a series of *draw string* procedures displays the results as shown in figure 3.

Odds & Ends

If all of this seems like the long way around to arrive at the shortest distance between two points, you are probably correct. In this example program with only 10 cities we could have as easily entered Latitudes and Longitudes in a form that would not require conversion. However, when considering there are over 6900 airports in the world, each with its own identifier and Latitude and Longitude, you can perhaps appreciate the simplicity of using whole numbers to represent degrees and minutes rather than complex numbers more easily assimilated by a computer.

One routine we didn't talk about this time is DoCloseBox. This routine is fairly straightforward, and performs the chores of disposing of the various windows and non modal dialogs used by the program. The only thing worth mentioning is my solution to avoid the possibility of opening the same window twice. In the DoOptions routine you will see that I have inserted a *disable menu item* procedure which globally disables the Options menu whenever a routine using a window is called. While this isn't the most elegant way of avoiding the embarrassment of not being able to close a window, it works. An *enable menu item* procedure in the DoCloseBox routine is executed to restore the Options menu after the current window is closed.

That about does it for this time. I hope what you have read here is helpful in learning Visual Interactive Programming. If you would like to see other V.I.P. stuff, write to MacTutor! In the meantime have fun, after all that's what it's all about.

```
byte
   AboutFlag
   AboutWindow
   City[4]
   EditDialog
   EditWindow
   FileName[7]={'C','i','t','i','e','s','\0'}
   InputDialog
   InputWindow
   Lat[7]
   Long[7]
   Menu[3]
   Quit
   RecordDialog
   RecordWindow
   WhichWindow
   Window
   flag
   go
   n

integer
   RecNo[10]

rectangle
   r

constants
   NoRecs = 10
   pi = 3.14159265

main
GC_Dist program for MacTutor™
by Bill Luckie  © 1987

Visual Interactive Programming
V.I.P. by Dominique Lienart, published by Mainstay.
==================================
DoSetup
while (! Quit)
   DoSelect
exit
==================================


Calculate (A,B,C,D,Origin,Destination)

-> real A
-> real B
-> real C
-> real D
-> byte Origin[4]
-> byte Destination[4]

byte
   S2[13]

integer
   Distance

real
   Dist
   fraction
   integer

rectangle
   r

assign (A / 100,A)
int & fract (A,integer,fraction)
assign (fraction / 60 * 100,fraction)
assign (integer + fraction,A)
assign (B / 100,B)
int & fract (B,integer,fraction)
assign (fraction / 60 * 100,fraction)
assign (integer + fraction,B)
assign (C / 100,C)
int & fract (C,integer,fraction)
assign (fraction / 60 * 100,fraction)
assign (integer + fraction,C)
assign (D / 100,D)
int & fract (D,integer,fraction)
assign (fraction / 60 * 100,fraction)
assign (integer + fraction,D)
assign (A * (pi/180),A)
assign (B * (pi/180),B)
assign (C * (pi/180),C)
assign (D * (pi/180),D)
assign (60*acos(sin(A)*sin(C)+cos(A)*cos(C)*cos(D-B)),Dist)
assign (Dist * (1/pi * 180),Dist)
int & fract (Dist,integer,fraction)
if (fraction ≥ .5)
   assign (integer + 1,Distance)
else
   assign (integer,Distance)
number to string ("0i",Distance,S2)
activate window (Window)
set text font (0)
move to (50,18)
draw string ("The distance from ",0)
draw string (Origin,0)
draw string (" to ",0)
draw string (Destination,0)
draw string (" is ",0)
draw string (S2,0)
draw string (" Nautical miles.",0)
==================================
DoAbout

byte
   ID
   go
   message
   type

point
   location

rectangle
   Portrect
   Windowrect

assign (0,go)
set rect (60,60,120,450,Windowrect)
set rect (0,0,120,450,Portrect)
new window (4,0,1,Windowrect,Portrect,"",Window)
set text font (0)
move to (15,85)
draw string ("GC_Dist program for MacTutor™",0)
move to (30,145)
draw string (" by Bill Luckie.",0)
move to (45,168)
draw string ("© 1987",0)
if (AboutFlag)
   while (! go)
      get next event (type,location,message,ID)
      if (type = 3)
         assign (1,go)
      else

else
   wait (90)
kill window (Window)
assign (1,AboutFlag)
assign (0,Window)
==================================
```


© The Definitive MacTutor, Vol. 4

```
DoCloseBox

get window (WhichWindow)
if (WhichWindow = AboutWindow)
   kill window (AboutWindow)
   assign (0,AboutWindow)
else
   if (WhichWindow = InputWindow)
      kill window (Window)
      assign (0,Window)
      kill dialog (InputDialog)
      assign (0,InputWindow)
      assign (0,InputDialog)
   else
      if (WhichWindow = EditWindow)
         kill dialog (EditDialog)
         assign (0,EditWindow)
         assign (0,EditDialog)
      else

enable menu item (Menu[3],0)
==================================
DoDialogEvent (Dialog,item)

-> byte Dialog
-> byte item

if (Dialog = RecordDialog)
   switch (item,1,2)
   case 1
      Get_Record
      Store_Records
   case 2
      kill dialog (RecordDialog)
      assign (0,RecordDialog)
   default

else
   if (Dialog = InputDialog & item = 1)
      Get_Input
   else
      if (Dialog = EditDialog)
         switch (item,1,2)
         case 1
            Next_Record
         case 2
            Save_Changes
         default

      else
==================================
DoEdit (item)

-> byte item

switch (item,3,4,5,6)
case 1
   cut text
case 2
   copy text
case 3
   paste text
case 4
   clear text
default
==================================
DoFile (item)

-> byte item

byte
   result

switch (item,1,2,4)
case 1
   set up page
case 2
   if (AboutWindow)
      print text (AboutWindow)
   else
      alert (1,"There is no window to print from.",result)
case 3
   assign (1,Quit)
default
==================================
DoMenuSelect (menu,item)

-> byte menu
-> byte item

switch (menu,Menu[1],Menu[2],Menu[3])
case 1
   DoFile (item)
case 2
   DoEdit (item)
case 3
   DoOptions (item)
default
==================================
DoOptions (item)

-> byte item

switch (item,1,2,3,5)
case 1
   Record_Dialog
case 2
   disable menu item (Menu[3],0)
   Edit_Dialog
case 3
   disable menu item (Menu[3],0)
   Input_Dialog
case 4
   disable menu item (Menu[3],0)
   Gee_Whiz
default
==================================
DoSelect

byte
   EventID
   EventMessage
   EventType

point
   MouseLocation

get next event (EventType,MouseLocation,EventMessage,EventID)
switch (EventType,1,4,5)
case 1
   if (EventMessage=0 & EventID=1)
      DoAbout
   else
      DoMenuSelect (EventMessage,EventID)
case 2
   DoCloseBox
case 3
   DoDialogEvent (EventMessage,EventID)
default
==================================
DoSetup

new menu ("File",Menu[1])
append menu item (Menu[1],"Page Setup...;Print...;(-;Quit/Q")
new menu ("Edit",Menu[2])
append menu item (Menu[2],"(Undo/Z;(-;Cut/X;Copy/C;Paste/
V;Clear")
new menu ("Options",Menu[3])
```

```
append menu item (Menu[3],"Create new Records...;View or Edit
Records...;Compute Distance...;(-;Gee Whiz Stuff...")
DoAbout
==================================
Edit_Dialog
rectangle
   r

set rect (100,106,250,406,r)
new dialog (r,"Edit Records",EditDialog)
set rect (111,23,144,116,r)
append dialog item (EditDialog,1,r,"Get Record")
set rect (111,179,144,280,r)
append dialog item (EditDialog,1,r,"Save Changes")
set rect (3,72,19,228,r)
append dialog item (EditDialog,4,r,"Review & Edit Records.")
set rect (27,50,43,150,r)
append dialog item (EditDialog,4,r,"City Code:")
set rect (47,50,63,150,r)
append dialog item (EditDialog,4,r,"Latitude:")
set rect (67,50,83,150,r)
append dialog item (EditDialog,4,r,"Longitude:")
set rect (87,50,103,156,r)
append dialog item (EditDialog,4,r,"Record No.:")
set rect (29,190,45,217,r)
append dialog item (EditDialog,5,r,"")
set rect (50,190,66,240,r)
append dialog item (EditDialog,5,r,"")
set rect (71,190,87,240,r)
append dialog item (EditDialog,5,r,"")
set rect (88,138,104,158,r)
append dialog item (EditDialog,5,r,"")
open dialog (EditDialog,EditWindow)
Review_Records
==================================
Find_City (Origin,Destination)

-> byte Origin[4]
-> byte Destination[4]

byte
   f
   flag1
   flag2
   match
   n
   result

integer
   RecNo[10]

real
   A
   B
   C
   D

compare string (Origin,Destination,match)
if (match = 0)
   alert (1,"The Origin and Destination cities are the
same.",result)
   return
else

assign (0,flag1)
assign (0,flag2)
open file (FileName,1,"TEXT",f)
for (n,1,NoRecs,1)
   allocate record (18,RecNo[n])
   read record (f,RecNo[n])
close file (f)
for (n,1,NoRecs,1)
   get field (n,0,4,City)
   upper case (City)
   compare string (City,Origin ,match)
   if (match = 0)
      get field (n,4,7,Lat)
      string to number (3,Lat,A)
      get field (n,11,7,Long)
      string to number (3,Long,B)
      assign (1,flag1)
   else

   compare string (City,Destination,match)
   if (match = 0)
      get field (n,4,7,Lat)
      string to number (3,Lat,C)
      get field (n,11,7,Long)
      string to number (3,Long,D)
      assign (1,flag2)
   else

if (! flag1)
   alert (1,"Sorry, Origin city code not in file.",result)
   return
else
   if (! flag2)
      alert (1,"Sorry, Destination city code not in
file.",result)
      return
   else

Calculate (A,B,C,D,Origin,Destination)
==================================
Gee_Whiz
byte
   AboutFile

rectangle
   PortRect
   WindowRect

set rect (50,40,312,472,WindowRect)
set rect (0,0,1000,432,PortRect)
new window (7,1,1,WindowRect,PortRect,"Gee Whiz
Stuff",AboutWindow)
open file ("Gee_Whiz",1,"TEXT",AboutFile)
load text (AboutFile,AboutWindow)
close file (AboutFile)
==================================
Get_Input
byte
   Destination[4]
   Origin[4]
   state

clear window (Window)
get dialog item (InputDialog,4,state,Origin)
upper case (Origin)
get dialog item (InputDialog,5,state,Destination)
upper case (Destination)
Find_City (Origin,Destination)
==================================
Get_Record
byte
   state

get dialog item (RecordDialog,7,state,City)
upper case (City)
get dialog item (RecordDialog,8,state,Lat)
get dialog item (RecordDialog,9,state,Long)
==================================
Input_Dialog
rectangle
   r
   r1

set rect (182,61,326,451,r)
```


© The Definitive MacTutor, Vol. 4

```
new dialog (r,"Select cities",InputDialog)
set rect (106,164,137,227,r)
append dialog item (InputDialog,1,r,"Do it")
set rect (31,104,47,204,r)
append dialog item (InputDialog,4,r,"Origin:")
set rect (62,104,78,204,r)
append dialog item (InputDialog,4,r,"Destination:")
set rect (34,220,50,245,r)
append dialog item (InputDialog,5,r,"")
set rect (60,220,76,245,r)
append dialog item (InputDialog,5,r,"")
set rect (3,49,26,342,r)
append dialog item (InputDialog,4,r,"Enter city codes for
Origin & Destination.")
set rect (52,68,152,444,r)
set rect (0,0,155,447,r1)
new window (4,0,1,r,r1,"",Window)
open dialog (InputDialog,InputWindow)
==================================
Next_Record
byte
   result
   string[13]

assign (n + 1,n)
get field (RecNo[n],0,4,City)
set dialog item (EditDialog,8,9,City)
get field (RecNo[n],4,7,Lat)
set dialog item (EditDialog,9,9,Lat)
get field (RecNo[n],11,7,Long)
set dialog item (EditDialog,10,9,Long)
number to string ("@b",n,string)
set dialog item (EditDialog,11,9,string)
if (n = NoRecs)
   assign (0,n)
else
==================================
Record_Dialog
rectangle
   r

set rect (100,154,251,359,r)
new dialog (r,"",RecordDialog)
set rect (115,5,147,85,r)
append dialog item (RecordDialog,1,r,"Save It")
set rect (115,119,147,199,r)
append dialog item (RecordDialog,1,r,"Quit")
set rect (4,38,22,176,r)
append dialog item (RecordDialog,4,r,"Enter new records")
set rect (30,6,46,106,r)
append dialog item (RecordDialog,4,r,"City Code:")
set rect (60,6,76,106,r)
append dialog item (RecordDialog,4,r,"Latitude:")
set rect (90,6,105,106,r)
append dialog item (RecordDialog,4,r,"Longitude:")
set rect (30,105,46,140,r)
append dialog item (RecordDialog,5,r,"")
set rect (60,105,76,162,r)
append dialog item (RecordDialog,5,r,"")
set rect (90,105,106,162,r)
append dialog item (RecordDialog,5,r,"")
open dialog (RecordDialog,RecordWindow)
==================================
Review_Records

byte
   f

integer
   result
open file (FileName,1,"TEXT",f)
for (n,1,NoRecs,1)
   allocate record (18,RecNo[n])
   read record (f,RecNo[n])
close file (f)
assign (0,n)
==================================
Save_Changes
byte
   f
   state
integer
   result

assign (0,flag)
open file (FileName,2,"TEXT",f)
if (n = 0)
   assign (10,n)
   assign (1,flag)
else

file position (f,(n-1) * 18,0,result)
get dialog item (EditDialog,8,state,City)
get dialog item (EditDialog,9,state,Lat)
get dialog item (EditDialog,10,state,Long)
allocate record (18,RecNo[n])
put field (RecNo[n],0,4,City)
put field (RecNo[n],4,7,Lat)
put field (RecNo[n],11,7,Long)
write record (f,RecNo[n])
close file (f)
if (flag)
   assign (0,n)
else
==================================
Store_Records
byte
   f
   ok
   result

integer
   Position
   RecNo[10]
   status

assign (n + 1,n)
if (flag)
   open file (FileName,2,"TEXT",f)
   get document length (FileName,Position)
   file position (f,Position,0,status)
   allocate record (18,RecNo[n])
   put field (RecNo[n],0,4,City)
   put field (RecNo[n],4,7,Lat)
   put field (RecNo[n],11,7,Long)
   write record (f,RecNo[n])
   free record (RecNo[n])
   close file (f)
else
   get document name (1,"Save file as
'Cities'","TEXT",ok,FileName)
   if (ok)
      open file (FileName,2,"TEXT",f)
      allocate record (18,RecNo[n])
      put field (RecNo[n],0,4,City)
      put field (RecNo[n],4,7,Lat)
      put field (RecNo[n],11,7,Long)
      write record (f,RecNo[n])
      free record (RecNo[n])
      close file (f)
      assign (1,flag)
      alert (1,"Additional records added during this session
will be saved without the GetFile dialog." ,result)
   else

set dialog item (RecordDialog,7,1,"")
set dialog item (RecordDialog,8,9,"")
set dialog item (RecordDialog,9,9,"")
```

Mac Hardware

MacBox and the Serial Sermon

Rev. Willis
MacFugNews
MacTutor Vol. 4 No. 5

Once upon a time in a galaxy far, far away...a humble servant of Big Byte took upon himself the serial expansion of the blessed Mac. However, before enlightenment was acheived, much wisdom had to be revealed unto and understood by the humble servant. In fact, much self-flagellation, usually with an RS-232 cable, was necessary to gain the wisdom needed to successfully expand the Mac's serial ports.

All hi-tech parables aside, my travail resulted from the desire to have my Mac talk to my modem, an X-10 Powerhouse appliance controller and a Radio Shack Model 100 portable computer, and do so without going through a cable swapping boog-a-loo. This was accomplished with an inexpensive switch and some simple wiring. I've included some pictures and diagrams to help the more ambitious (and cheap, Mac switch boxes are commercially available) of you duplicate the task.

Excluding the SCSI port and the second disc drive port, the Mac talks to the world through two serial ports. One of these serial ports is usually configured through the Chooser software to talk to the Imagewriter. The other port can be configured by Chooser as an Appletalk connection, or, using communications software such as Red Ryder, as a modem port. Also, using an adaptor box and MIDI software, these serial ports can talk to musical instruments such as a DX-7 sythesizer. Finally, one of these ports can be used to talk directly to another computer without the modem/phone line connection assuming both machines are in close proximity. Some computers, such as the Incredibly Bad Machine and the Model 100, normally talk to printers through a parallel port; however, the Imagewriter/Mac combo do it serially.

The RS-232 data communications standard is used by most personal computer systems for data transfer. There is also Ethernet, Appletalk and others, but we're talking proletarian stuff here, not big-time University networking. The Mac uses an improvement of the RS-232 standard called RS-422. This improvement allows for longer cable lengths to carry digital data, necessary for Appletalk networks. Thankfully, RS-422 and RS-232 protocols are generally compatible. The RS-232 standard defines the DB-25 connector pinouts (check the diagrams included) as well as describes data terminal equipment (DTE) sending digital information to data communications equipment (DCE) over an analog phone line to another DCE-DTE combo (digital data poops out after 100 feet or so and must be converted to an analog signal for the long haul). DTEs are usually computers, while DCEs are modems. They are connected with straight cables, that is, the TXD pin on the DTE is connected to the same TXD pin on the DCE. The internal wiring of the equipment makes sure everthing is copesetic. The DTE/DCE pairs use software handshaking to control the communications

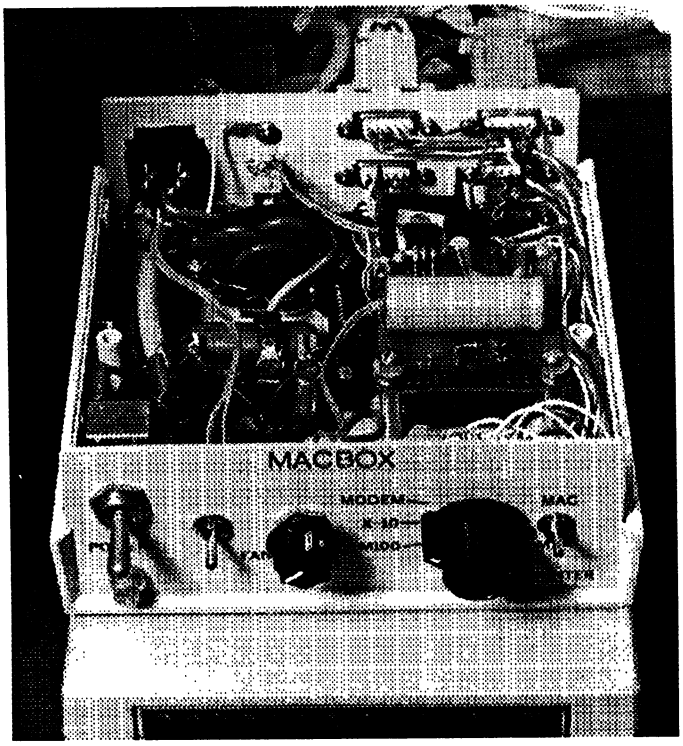


Fig. 1 TIFF Photo of the MacBox Internals

connection, usually XON/XOFF protocol. If one pair wants to pause communications, it issues a little ASCII data code ,Control Q, to tell the other pair to stop transmitting; a Control S resumes communications. If you ever used Red Ryder to access Compu$erve or Genie, you may have used the Q and S radio buttons to control the information zipping across the screen at a rate faster than you can read it.

However, as occurs with all good standards, they get modified to accomplish new tasks. A problem arises when two computers are hooked back-to-back with no modem/phone link inbetween. Now you have a DTE-DTE link which confuses the signal lines on the interconnection cable. So you need a cross or "null" cable to connect the equipment. In such a cable, the TXD pin from one DTE goes to the RXD pin of the other DTE, and vice versa. If these DTE-DTE combos are a back-to-back computer pair, they usually use a form of software handshaking. However, if a computer is talking to a printer (which usually acts as a DTE), then they may use a form of hardware handshaking: two control lines are crossed and the pins toggled with a logic "Hi" to indicate the DTEs are ready to communicate. If one DTE toggles its control pin "Lo", communications are paused until the pin is again toggled "Hi". The control lines usually link the RTS-CTS or DTR-DSR pins. In the case of the Mac and the Imagewriter, the handshaking lines are CTS-DTR. If the IMW needs to pause,

© The Definitive MacTutor, Vol. 4

the "Hi" on its DTR pin goes "Lo"; the Mac detects this new condition on its CTS pin and stops sending data until its CTS pin is toggled "Hi" again. Got it?

Another problem, and a further bastardization of the RS-232 standard, occurs when two DTEs want to talk to each other using software (XON/XOFF) handshaking but have pins on their serial ports that are expecting hardware handshaking. This situation requires the old "fake-out" handshaking method. Pins that are normally "Hi" are jumpered to the pins that expect a "Hi" before they will allow data transmission to begin. RS-232 cables for the IBM are notorious for this and require jumpering several pins to get the serial port to work. Usually, the normally "Hi" RTS pin is jumpered to the normally "Lo" CTS pin, while the normally "HI" DTR pin is jumpered to both DSR and CD. This "fake-out" method is very common in situations where multi-conductor cable is not available or where the exact method of hardware handshaking is not known. All in all, the RS-232 standard has become a very unstandard standard.

Anyway, enough sermonizing and on to some constuction details for the ultimate in Mac serial port switching...MACBOX!!! The first detail that needed to be resolved was the 8-pin miniDIN socket used by the newer Macs. The required plugs are impossible to find and wouldn't work well panel mounted into a box. Since my Mac was an updated 512 machine, I had a couple miniDIN to DB-9 adaptor cables that I extended to reach MacBox. Panel mount DB-9s are readily available, so I used two male DB-9s as inputs from the Mac and four female DB-9s as outputs to peripherals. Note when wiring the pins that they are numbered in opposite directions on the male and female connectors (I didn't). The protective and signal grounds are now tied together within the Mac, so only three wires are normally needed to connect serial devices. I included a handshaking line since it was wired that way in my modem cable and is needed for the IMW (the other devices handshake through software XON/XOFF).

CONNECTOR PINOUTS

Mac miniDIN RS-422		Mac DB-9 RS-422		Peripheral DB-25 RS-232	
DTR	1 * Output H/S	DTR	6 *	DTR	20 * Data Terminal Ready (DTE output)
CTS	2 * Input H/S	CTS	7 *	CTS	5 * Clear to Send (DTE input)
TXD-	3 * Tx Data	TXD-	5 *	TXD	2 * Transmit Data (DTE output)
SG	4 * Signal Ground	SG	3 *	SG	7 * Signal Ground
RXD-	5 * Rx Data	RXD-	9 *	RXD	3 * Receive Data (DTE input)
TXD+	6 *	TXD+	4 *	RTS	4 * Request to Send (DTE output)
N/C	7 *	N/C	2 *	DSR	6 * Data Set Ready (DTE input)
RXD+	8 *	RXD+	8 *	DCD	8 * Data Carrier Detect
		PG	1 *	PG	1 * Protective Ground

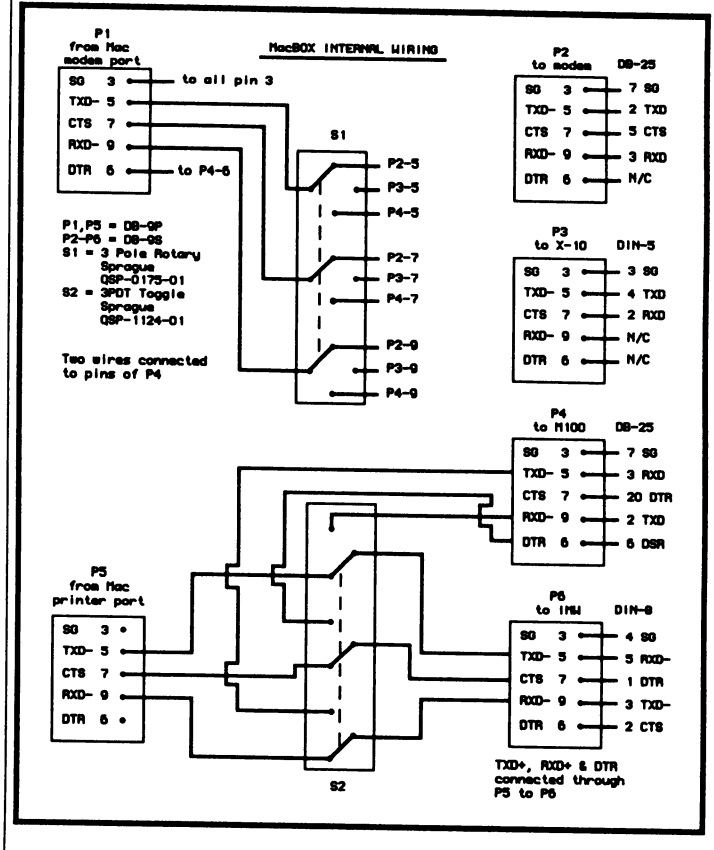
Mac no longer uses PG (internally jumpered to SG)
Mac uses DTR/CTS hardware handshaking to IMW
TXD+ and RXD+ used in RS-422 communications (Appletalk)
RTS, DCD and DSR not normally used by Mac

Initially everything worked great until I decided to get the M100 to talk to the IMW. First of all, the M100 usually talks to printers through a parallel port. Thankfully, the built-in BASIC accesses the serial port. Also, I have SuperRom installed in my M100 which contains a word processor that accesses the serial port. So, another switch and a little more wiring later, I was soon printing away on the IMW from the M100...sort of. After about a page of data, everything would turn to garbage. Obviously I was overloading the IMW's input buffer, no amount of toggling the IMW's DTR pin was shutting off the M100 from continuing to send data. I soon discovered that while the handshaking pins were active on the M100, there was no firmware (ROM routines) to recognize hardware handshaking. About then Big Byte stepped in and handed me a set of golden tablets containing a M100 mod that would give the M100 adequate hardware handshaking. The mod details involved the piggybacking of a quad nand gate chip inside the M100 to use the DSR line to read the IMW's DTR line. To be more consistent to the Mac's way of doing things, I could have easily use the CTS line instead. (See references below)

Soon the M100-IMW connection was yakking away with no problems...except now the M100 wasn't talking to the Mac. I soon realized that I needed to run another wire from the Mac's DTR pin to the M100 DSR pin to "fake out" the M100 even though it was talking to the Mac using software handshaking. Tracing the wiring in the accompanying diagram should reveal the necessary connections.

References:

MacAccess, Hayden Books,1987
Understanding Data Com., TI Library
Imagewriter II Owner's Manual
Portable 100 Magazine, 12/87
Modern Electronics Magazine, 5/86



Resource Roundup

Comments About PICTs

Joel West
Palomar Software, Inc.
MacTutor Contributing Editor
MacTutor Vol. 4 No. 6

*Joel West is the author of the best selling MPW book titled **Programming with Macintosh Programmer's Workshop**. He is also the principle in Palomar Software, which produced the popular **Colorizer DA** and recently, the **Pict Detective** utility. In this column he continues his Resource Roundup with some expert insights into PICT resources.*

PICTs

This month's focus will be on PICT resources, handles and documents. These QuickDraw pictures are the *lingua franca* of Macintosh graphics, the common ground between diverse applications.

As I'm always fond of emphasizing, software developers (yes, that means you!) must support compatibility and standards to keep the users and marketing types happy. Whatever other graphics formats you prefer, the standard user interface requires you to support PICTs in some form or another.

Besides, PICTs are the native format of QuickDraw, a mere transcription of the various QD calls. In practical terms, PICTs are what you get for free with your Macintosh graphics, so if you can display it on the screen, you can record it in a PICT.

Now while I'm big on standards, I'm not so dogmatic as to ignore the reality that PICTs are often the lowest common denominator, such as when you're writing a complex PostScript-based program. Still, that doesn't mean you should ignore PICTs, or make them look like heck—since that's also what you'll get on the many non-PostScript printers.

What kind are there?

Our Mac world is divided into two classes of people: those own Macintosh II's, and the impoverished masses. (Before anyone starts flaming, let me say that I'm writing this with the SE on my desk at Palomar, and with my Plus at home, since other people are using my Mac II.)

Anyway, there are the original, classic, or "PICT-1" pictures that were defined with the Macintosh 128, and unchanged by the Plus and SE. Any Macintosh can generate these and display them accurately.

The Macintosh II can also generate and display new, color, or "PICT-2" pictures that require Color QuickDraw. The main differences are that you get more than 8 colors, and you can have color bit images.

You can use a PICT-2 on a Plus or an SE, thanks to a clever DrawPicture() patch in System 4.1 and later. (See "Sleuthing the New System File" in the August 1987 edition of *Resource Roundup*.). The patch draws a PICT-1 approximation of the color picture. Unfortunately, in doing so, the patch throws all the interesting information on the floor.

There is a reasonable way to preserve this information (Palomar would be glad to provide it to any developer upon request), but interest in doing this seems to be underwhelming. My conclusion is that we will have to live with this lobotimized approximation of PICT-2's for as long as applications must support machines without Color QuickDraw. (Three years? Five years?) *[At the recent Developer Conference, MacTutor asked the Apple guru's why Apple didn't simply create one version of quickdraw for all machines so developers would not have to support multiple window, menu and PICT data structures. The answer was "Quickdraw requires the 68020". Now I know that! But they could have written a 68000 version for the SE that allowed color quickdraw calls which defaulted to black and white, presenting to the developer a single programming model. It appears Apple intends to upgrade the SE to a 68020 and solve the problem that way. Incidently, there was some controversey over letting me ask that question, as the panel moderator said I was "Press", a dirty word at Apple, I guess. I would like to thank all the friends of MacTutor who hooted down the Apple representative so I could ask my question. -Ed]*

Where do you find them?

QuickDraw pictures are usually found in one of three places:

- In a resource of type 'PICT';
- In the clipboard; or
- In the data fork of a file of type 'PICT'.

All three usages can be found in a typical PICT-oriented

MacTutor
The Macintosh Programming Journal

Fig. 1 Our sample PICT object parsed with Pict Detective

© The Definitive MacTutor, Vol. 4

application. Resources are created when the user copies a PICT into the Scrapbook; they're also used by applications that need to display a PICT in a dialog, since one of the predefined dialog item types is a PICT resource.

The other two formats are directly employed by users to interchange graphics between two programs. As documented with the Scrap Manager, the standard user interface requires all programs to be able to import a 'PICT' clipboard entry, while all but the most brain-damaged of graphics programs can also open and save PICT files.

How do you get one?

Finding a PICT resource is criminally easy; depending on what you're up to, you're likely to try

```
ph: PicHandle;

ph := PicHandle(
    GetResource('PICT', resno));
FailNIL(ph);
```

or maybe

```
ph := PicHandle(
    Get1IndResource('PICT', 1));
FailNIL(ph);
```

(All examples shown here are in MPW Pascal, with a MacAppish flavor. I find myself re-implementing functions like FailOSErr and FailNIL for my non-MacApp programs, just to keep consistent.)

The clipboard is a pretty easy place to grab a picture, too:

```
ph: PicHandle;
len: LONGINT;

ph := NewHandle(0);
len := GetScrap(Handle(ph),
    'PICT', offset);
IF len < 0 THEN
BEGIN
    IF len = noTypeErr THEN
        (* there wasn't one *)
    ELSE
        FailOSErr(OSErr(len));
    EXIT(ThisRoutine);
END;
```

Reading a PICT from the data fork of a file is also child's play. Like most of Apple's early formats, the first 512 bytes are a header; for now, we ignore them, but if you really care, the format is described in *Macintosh Technical Note #27*, "MacDraw's PICT File Format and Comments."

Anyway, this is all it takes to figure how long the picture is and read it into a new handle:

```
FailOSErr(GetEOF(docrefno, filelen));
FailOSErr(SetFPos(docrefno,
        fsFromStart, 512));

piclen := filelen - 512;
phand := NewPermHandle(piclen);
FailNIL(phand);

HLock(phand);
FailOSErr(FSRead(docrefno,
        piclen, phand^));
HUnlock(phand);
```

Of course, I'm oversimplifying—or, as Kurt Schmucker used to say in his courses, "I'm lying to you just a little." This all assumes that you have lots of memory (always a bad assumption) and that every picture is small (even worse.) More on that in a minute. Suffice it to say that, if you have large pictures and not much memory, the above code will give up.

What do you do with them?

Each picture has one piece of useful information directly readable by any application: its bounding rectangle. This is read by the code

```
arect := ph^^.picFrame;
```

The original documentation also says there's a size in the picture (picSize), but that's a big lie, not a small one. Nowadays, there are lots of big pictures, those longer than the 32,767 limit of a signed 16-bit integer. For reliable results, you must get the size from some other means (if you don't have it already), such as from the Memory Manager:

```
piclen := GetHandleSize(ph);
```

Of course, there's really only one good thing to do with a picture: draw it. This is probably the simplest thing you can do with a picture:

```
DrawPicture(ph);
```

(This is another little lie, since QuickDraw can fail and not tell you about it, particularly with a big picture, but finding out whether QuickDraw is failing is beyond the scope of this article.)

Of course, you can also invert all the previous steps for finding a picture: create a new resource, write the PICT to the clipboard, or write it to the data fork.

Creating a new resource is a matter of two calls,

```
resid := Unique1ID('PICT');
AddResource(ph, 'PICT', resid, '');
FailOSErr(ResError);
```

Writing it to the clipboard is also pretty easy:

```
FailOSErr(ZeroScrap);
HLock(ph);
FailOSErr(PutScrap(piclen,
        'PICT', ph^));
HUnlock(ph);
```

Writing it to disk is also easy; just don't forget to zero the 512-byte header if you don't have something interesting to put in

the header (a reasonable simplifying assumption). Shown below is also one of the ugliest ways to generate the zero'd header, but is included to keep the code easy to follow:

```
FailOSErr(SetEOF(docrefno,
        piclen+512));
long := 0;
FOR i:=1 TO 128 DO
    FailOSErr(FSWrite(docrefno,
        SIZEOF(LONGINT), @long));

HLock(phand);
FailOSErr(FSWrite(docrefno,
        piclen, phand^));
HUnlock(phand);
```

PICT Sleuthing

Regular readers of *Resource Roundup* will recall previous articles that talked about keyboard sleuthing, printer sleuthing, and system file sleuthing.

Unfortunately, the format of pictures is so complicated that sleuthing an arbitrary picture a byte at a time is a very involved process. Just when I thought I had a handle on it, along came PICT-2, making it even worse.

At one point, I'd hoped to finish my program by August 1987, but that was not to be. Instead, a local Mac developer, Marshall Clow, finished up the work and made a product out of something that I'd started, so we could release PICT Detective to the world at January's Expo in San Francisco. The response has been suprisingly strong for such a specialized tool—proof positive how many developers are doing graphics development.

PICT Detective™ is a set of utility programs for the MPW programs. It consists of three MPW tools:

- DePict decompiles PICT resources and documents into Rez source form
- PictFork moves a PICT from a resource to a PICT document, or vice versa.
- ShowPict puts up a modal dialog to allow you to peek at the PICT while within MPW.

In addition, Pict.r contains enough declarations to be able to recompile any DePict'd data back to a picture. An earlier version was included in Chapter 9 of *Programming with Macintosh Programmer's Workshop*, but it has two severe limitations: it fails badly with bitmaps, and it doesn't support the PICT-2 format.

You can also fiddle with the source before recompiling, or define new pictures completely from scratch using the Rez source form. Of course, Rez only compiles to the resource fork, but PictFork will move the compiled picture back to the data fork.

For the duration of this article, I'll be illustrating the contents of PICTs using the DePict format, the only form I know of today for describing pictures.

To give you some idea of what the output can look like, first take a look at the picture shown in Figure 1, which was created using SuperPaint. Now look at is its description of a simple picture in DePict form:

```
resource 'PICT' (0, purgeable) {
194, /* Actual length: 194 */
(0, 0, 720, 576),
(   Version( 1);
    ShortComment( picDwgBeg );
    ClipRgn( 10, (0, 0, 720, 576),
            $"");
    PnPat( $"88 00 22 00 88 00 22 00");
    OvSize( (18, 18));
    PaintRRect( (57, 40, 133, 425));
    PnSize( (4, 4));
    PnPat( $"FF FF FF FF FF FF FF FF");
    FrameSameRRect( );
    LongComment( picTextBegin(
            6, 0, 0, 2, 0x00));
    LongComment( picTextCenter(
            0xFFF40000, 0x00608000));
    ShortComment( picStringBegin );
    TxFont( times);
    TxSize( 48);
    DHDVText( -120, 102, "MacTutor");
    ShortComment( picStringEnd );
    ShortComment( picTextEnd );
    LongComment( picTextBegin(
            6, 0, 0, 2, 0x00));
    LongComment( picTextCenter(
            0xFFFC0000, 0x00628000));
    ShortComment( picStringBegin );
    TxSize( 13);
    LongText( (116, 134),
    "The Macintosh Programming Journal");
    ShortComment( picStringEnd );
    ShortComment( picTextEnd );
    ShortComment( picDwgEnd );
    EndOfPicture( )
)
};
```

Note that the length shown is the 16-bit picSize length, while the "actual length" is the length of the resource—usually the same for resources less than 32K long.

Otherwise, if you know QuickDraw, the description is nearly self-explanatory. About the only opcodes that might not be familiar are ShortComment and LongComment, which brings us to...

PICT Comments

Lots of the interesting stuff in a picture won't fit in QuickDraw's mindset. Some of this stuff doesn't belong in QuickDraw, other probably did belong, but didn't make it in time.

But there is a portable way for applications to share meta-information with other code: via picture comments. These are portions of a picture that are recorded and replayed by QuickDraw, but, by default, have no effect on the displayed image.

Picture comments have three principal uses:

- To convey additional information that is useful for an application that is reading a picture previously saved to the clipboard or a document. An example of this is grouping objects, alá MacDraw.
- To communicate information from an application to the current printer driver. An example of this is including PostScript output for the LaserWriter.

- To get around some implementation limitation in QuickDraw. This was particularly severe with the stack-based algorithms of the 64K ROM (which worked around the puny heap of a 128K Mac, but prone to stack overflow), but still required for large bit images.

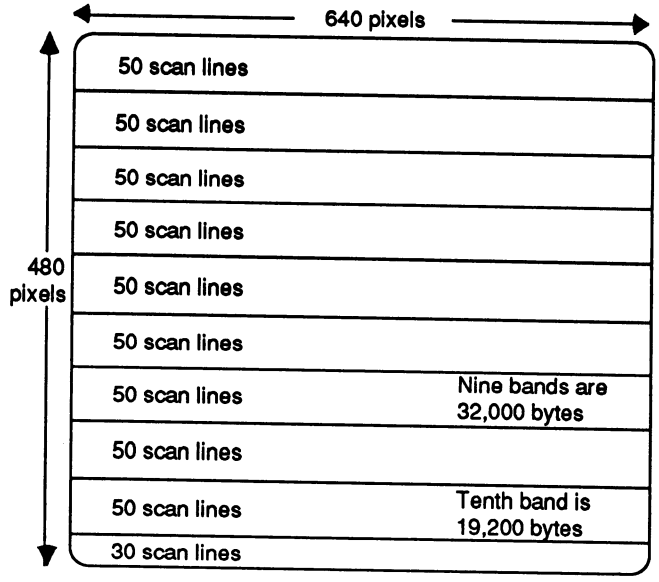
Of course, there's an overlap between these categories. Printer drivers can ignore some of the comments, but applications typically generate (and must recognize) all three categories.

Table 1 shows a list of the most common and important picture comments. Again, the names are those used by DePict; the main difference is that any comment that didn't start with pic had it added as a prefix, for consistency's sake and to make it easy to distinguish picture comments from opcodes.

Banding Bits

When you draw a bit image on the screen with CopyBits(), QuickDraw uses roughly the same amount of memory as your bitmap (or pixmap) as a temporary scratch memory. This is sometimes too much for your Mac to tolerate.

Clever programs break larger bit images into a series of horizontal bands, as illustrated by Figure 2. Each band is drawn with a separate CopyBits(), one below another, until the entire image is covered.



Macintosh II screen PixMap (640x480x256 colors)

Fig. 2 PixMap

The picBitBeg and picBitEnd comments indicate to an aware application that the intervening bits are bands of a single image, not separate images. Your code looks something like:

```
PicComment(picBitBeg, 0, NIL);
FOR i := 1 TO nbands DO
BEGIN
    ...
    CopyBits(...);
END;
PicComment(picBitEnd, 0, NIL);
```

This is one case where an application *must* understand an input picture comment. Otherwise, your user will see a series of separate images from any program that is kind enough to band the data into reasonable-size pieces.

At Palomar, we live by the gospel we preach. When it came time to do the Colorizer package for the Mac II, we made sure that the Color SaveScreen™ screen dump FKEY outputs bands. The Colorizer application recognizes a series of bands between picBitBeg and picBitEnd as being a single bitmap object.

How small should thou makest thine bands? Once upon a time, in the days of olden ROM, very tiny 3.5K bands were recommended. Thou must forsake this ancient ritual, since thou shalt fail miserably with the images of Color QuickDraw, which art 300K or more.

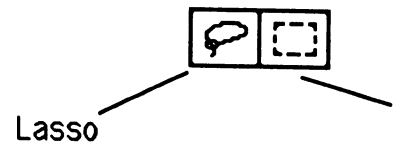
As long as you're stick with a Macintosh Plus or later, a size of 20K-40K seems about right, according to Apple technoid, and MacTutor Founding Board Member, Chris Derossi, who's researched the subject. Some people like 32,768 (or slightly less). I happen to insist on a minimum of 21,888, so that a standard-screen Plus or SE screen dump is never banded.

Note that the sizes measure the size of the bit image in memory, or height*width*depth/8. The actual size in the picture will be somewhat less, due to the PackBits compression performed by QuickDraw when recording the picture.

Realize that the larger the band size, the more temporary memory will be necessary to display your picture. On the other hand, the smaller the band size, the more bands you have; because each band has a fixed overhead, this can increase the size of your picture. For example, a 256-color PixMap has an overhead of about 1,100 bytes for each band; if memory serves me, Chris Derossi was able to save about 30K out of a 300K image by increasing the 20K bands to 40K.

Whoa, Trigger!

While we're on the subject of bits, I'd also like to remark on a truly under-documented picture comment, picLasso. How did the original MacPaint tell the difference between a bitmap selected with the rectangular selection tool and one selected with the lasso? (The tools shown below.)



With a picture comment, of course. This same picture comment is also used by HyperCard and SuperPaint, and probably some other applications I don't know about.

I went ahead and selected a small image in SuperPaint and put it into the Scrapbook. This is what it looked like in DePict form:

```
resource 'PICT' (-32766, purgeable) {
93, /* Actual length: 93 */
(0, 0, 14, 16),
```

```
(   Version( 1);
    ShortComment( picDwgBeg );
    ShortComment( picBitBeg );
    ClipRgn( 10, (0, 0, 720, 576), $"");
    BitsRect(
        2,
        (0, 0, 14, 16),
        (0, 0, 14, 16),
        (0, 0, 14, 16),
        srcOr,
            $"03F8"
            $"1C06"
            $"2001"
            $"4001"
            $"8001"
            $"8006"
            $"8038"
            $"71C0"
            $"CE00"
            $"A800"
            $"7000"
            $"1000"
            $"1000"
            $"2000"
        );
    ShortComment( picBitEnd );
    ShortComment( picDwgEnd );
    EndOfPicture( )
)
);
```

Now, if the same image is lasso'd, there is one main difference. The first few opcodes include the picLasso comment, as in:

```
Version( 1);
ShortComment( picDwgBeg );
ShortComment( picLasso );
ShortComment( picBitBeg );
ClipRgn( 10, (0, 0, 720, 576), $"");
BitsRect(    /* etc. etc. */
```

Tallying Text

When you're including text in a drawing, a group of comments allow you to both exchange data with other applications and take full advantage of the high-resolution LaserWriter text placement.

The picTextBegin comment allows including the following information:

- Left, right, center or full justification
- Rotation of text, 0-360° (although usually a multiple of 90°)
- Flipping coordinate system along horizontal or vertical axis
- Single, 1.5 or double spacing

This affects all the text objects up until the picTextEnd comment. The picTextCenter comment specifies the center of rotation for this group of text.

If you want to see more code demonstrating these picture comments, look at "Editing Rotated Text" by John D. Olsen in the February 1988 issue of *MacTutor*. The picTextBegin comment is also important when it comes to talking about application compatibility with printer drivers.

Incidentally, the old ROM also required banding pieces of strings with picStringBegin and picStringEnd. On reading a commented PICT, you should strip out these archaic comments and reassemble into a single string. On output, don't bother to put them in.

Pacifying MacDraw

MacDraw has its own set of picture comments. Some of them are useful to other graphics programs, or to printer drivers; others are just odd MacDrawisms.

The first such comment is picDwgBeg. Don't ask me why, but MacDraw wants to see this comment. However, it *must* be the very thing you add to the picture, and it must be closed with picDwgEnd as the very last thing you put in the picture. For example,

```
ph := OpenPicture(somerect)
PicComment(picDwgBeg, 0, NIL);
(* other drawing here ... *)
PicComment(picDwgEnd, 0, NIL);
ClosePicture;
```

More useful information is contained in the grouping comments picDwgBeg and picDwgEnd, which bracket objects that have been grouped together to act like one object. This concept (and the associated comments) is useful for any graphics program, which is why we made it one of the few comments supported by the initial version of our Colorizer application.

Other comments are related to arrowheads on lines. MacDraw provides arrowhead information in a PICT two ways. For it own use, it includes picture comments indicating what sort of arrowhead should be drawn on the beginning or end of the line, or both. For other programs, it includes a screen representation of the arrowheads, as in

```
ShortComment( picArrw1 );
ClipRgn( 10, (0, 0, 720, 540), $"");
PaintArc(    (35, 288, 55, 308), 246, 48);
Line(    (45, 36), (45, 289));
ShortComment( picArrwEnd );
```

MacDraw II has fancier arrowheads, but I haven't laid my hands on a copy yet, so I can say how it represents those arrowheads in a PICT.

Finally, MacDraw has a really *wierd* way of defining polygons. So wierd, in fact, that many third-party applications receive polygons from MacDraw and don't realize it.

Instead of using QuickDraw polygon calls, MacDraw represents all polygons by bracketing a series of line segments with picPolyBegin and picPolyEnd picture comments. A simple MacDraw polygon looks like:

```
resource 'PICT' (0, purgeable) {
60,  /* Actual length: 60 */
(0, 0, 720, 540),
(   Version( 1);
    ShortComment( picDwgBeg );
    ShortComment( picPolyBegin );
```

```
        ShortComment( picPlyClo );
        ClipRgn( 10, (0, 0, 720, 540), $"");
        Line( (27, 45), (135, 288));
        ShortLineFrom( -126, 36);
        ShortLineFrom( -99, -63);
        ShortLineFrom( 0, -9);
        ShortLineFrom( -18, -72);
        ShortComment( picPolyEnd );
        ShortComment( picDwgEnd );
        EndOfPicture( )
    }
    };
```

Roll Your Own Comment

Finally, there is a portable way to define your own picture comments and include information private to your application. You might want to do this when copying an image to the clipboard, if your application wanted to keep some information beyond the standard QuickDraw definition.

As described in *Macintosh Technical Note #181*, "Every Picture [Comment] Tells Its Story," the new picAppComment includes the application signature and a variable amount of data. Since every good developer registers his/her unique application signature with Macintosh Developer Technical Support, this means there will never be a conflict with other applications, right?

I included an example of using this application-specific comment in the PICT Detective manual. To my knowledge, none of the software released as of this writing uses this comment, but such software will definitely be hitting the streets later in 1988.

P.S. on PostScript

In a future article, I'll talk a little more about how picture comments are used to communicate between an application and a printer driver.

Many of the comments that are defined have to do with PostScript. Do you want to output direct PostScript to a printer? You have a lot of choices on how to throw it into a picture comment.

However, one comment is so standard that even a cursory discussion of comments would be incomplete without it.

The comment, picPostScriptHandle, is also one of the easiest to use: the remainder of the comment is nothing more than raw PostScript. When you see it in DePict output, the name seems a little odd, but it makes sense when you see the code to generate it.

To take an extremely trivial example, suppose you want to precede any of your custom PostScript with your own proprietary notice, defined by a string resource:

```
resource 'STR ' (1024) {
"%Copyright 1988 FooBar Software");
```

To add this to your PostScript output, you merely need the code:

```
hand: Handle;
siz: LONGINT;
hand := Handle(GetString(1024));
siz := GetHandleSize(hand);
PicComment(picPostScriptHandle,
    siz, hand);
```

Later, you might take the picture file and disassemble it with DePict. The data would look something like:

```
resource 'PICT' (0, purgeable) {
4062,
{0, 0, 342, 512},
{   Version( 1);
    ClipRgn( 10, (0, 0, 342, 512),
        $""),
    LongComment(
        picPostScriptHandle(
    "%Copyright 1988 FooBar Software")};
...
```

Finding PICT Comments

As noted before, analyzing a picture a byte at a time is very complicated and requires that you build a complete parser similar to Apple's.

The *simple* way is to let QuickDraw do the parsing for you, and then have your own code called just for what you want to look at.

All you have to do is install your own commentProc in the bottleneck procs, as in:

```
qdp := NewPtr(SIZEOF(QDProcs));
SetStdProcs(qdp^);
    (* set default values *)
qdp^.commentProc := @MyComment;
    (* install our interpreting routine *)
grafPort^.grafProcs := qdp;
```

The calling sequence for your comment-interpreting proc is the same as for the StdComment procedure:

```
PROCEDURE MyComment(kind,
                    dataSize: INTEGER;
                    dataHandle: Handle);
BEGIN
    CASE kind OF
    ...
    END
END; {MyComment}
```

As you can see, this is just the inverse of the PicComment() call, with each of the parameters having the same significance.

While you're writing your own comment-reading routine, you might as well include the call

```
StdComment(kind, dataSize, dataHandle);
```

at the end of your code, just in case. You have no way of knowing what else has been installed there, but it's possible some future system software might patch this to catch the sorts of things currently not supported by QuickDraw pictures.

The same process of installing your code in grafProcs is also

used if you must display or record larger pictures than can be stored in memory at one time. The getPicProc and putPicProc fields redefine what's done by DrawPicture() and OpenPicture(), respectively. The correct use of spooling pictures like this is left for another time.

Conclusion

PICT resources are the standard format for Macintosh graphics, and, as such, QuickDraw provides tools that make it easier for your application to use PICTs.

The PICT format can also be expanded beyond QuickDraw's definition via the use of picture comments. These can be used for portably exchanging data between applications, or for storing information specific to a particular application.

If you want to know more about picture comments, see:

- *Macintosh Technical Note #27* (mentioned above)
- *Macintosh Technical Note #91*, "Optimizing for the LaserWriter—Picture Comments"
- *Macintosh Technical Note #181* (mentioned above)
- *LaserWriter Reference* (APDA, 1987)

[At the printing session of the Developer Conference, Apple reps made the strange comment that they wished the PicComment would go away. Yet they had no alternative in mind to how Quickdraw limitations could be circumvented without PicComments. They also denied any interest in Display Postscript. Finally, they had no comment on any future Apple enhancements for Quickdraw that would provide an alternative to PicComments. The print gurus inside Apple seem to be saying we should all use normal quickdraw so Apple can continue it's printer independence myth, don't use PicComments, and forget that Quickdraw by itself is inadequate for today's sophisticated graphics requirements. We find this either a "head in the sand" attitude or Apple is working on something they don't want us to know about. Other Apple reps, when asked about this, said they may wish PicComments would go away, but they won't because they are necessary and everything would break if they did. -Ed]

Figure 1: Simple PICT

Figure 2: Banded bits

Figure 3: MacPaint selection tools

Num	Name	Description
100	picAppComment	Application-specific comment
130	picDwgBeg	Beginning and end
131	picDwgEnd	of a MacDraw picture
140	picGrpBeg	Beginning and end
141	picGrpEnd	of grouped objects
142	picBitBeg	Beginning and end of
143	picBitEnd	a banded bitmap
150	picTextBegin	Beginning of text string Specify text rotation
151	pictTextEnd	End of a text string
154	picTextCenter	Specify center of rotation
152	picStringBegin	Beginning and end of
153	picStringEnd	a banded string (archaic)
160	picPolyBegin	Beginning and end of a
161	picPolyEnd	MacDraw-style polygon
170	picArrw1	Arrowhead on 2nd point of line
171	picArrw2	Arrowhead on 1nd point of line
172	picArrw3	Arrowhead on both endpoints
173	picArrwEnd	End of arrowhead comment
192	picPostScriptHandle	PostScript picture comments
12345	picLassoSelected	bitmap is lasso'd

Table 1: Application picture comments

Resource Roundup

System Sleuthing II: The Sequel

Joel West
Palomar Software, Inc.
MacTutor Contributing Editor
MacTutor Vol. 4 No. 7

DePict

About this time last year, I wrote an installment of *Resource Roundup* on the various changes in trap patches from System 3.2 to 4.1, and the compatibility issues related to the use of those systems ("Sleuthing the New System File," August 1987.)

That article seemed to have struck a chord, with many programmers (including some generous folks from Apple) telling me it was extremely valuable to have the information all pulled together all in one place.

Since that time, the original story has continued where we left off last time, with two new system software releases introduced to date.

So, like many Hollywood writers and producers lacking creativity, I've decided to go for the easy buck, the cheap knock-off; in short, a sequel:

Sleuthing the System File,Part II: MultiFinder.
Starring System 4.2 (née 5.0) and 6.0.
Written by yours truly.
Produced by Apple Computer, Inc.

Incidentally, we'll pick up the story where we left off last time, so if you have the earlier article handy, you may wish to grab it to refresh your memory.

A Number of Numbers

First, let me clear up a little appelative confusion in these most recent systems.

I have two sets of disks sitting next to my desk. One is marked 5.0, dated October 1987; another is marked 6.0, dated April 1988. However, 5.0 is not called 5.0 by most of its components, although 6.0 is much better in this regards.

If you'll recall our last episode, the naming scheme is somewhat arcane for the System and Finder (and now MultiFinder). Table 1 shows an updated list of software versions, and the systems with which they are compatible.

Release 5.0 actually consists of System 4.2, Finder 6.0 and MultiFinder 1.0 (figure that one out!). However, Release 6.0 offers an important step towards sanity, with version numbers 6.0, 6.1, and 6.0, respectively. It seems apparent that all the numbers will eventually be the same.

Let's hope they stop changing the leading digit every 6 months, or we'll be up to version 19.0 (instead of 9.3) in no time at all. I also like a two-part rule-of-thumb I've heard for software versions that is not specific to Apple's software: 1. When they change the first digit, the programmers were ambitiously adding new features, so watch out! 2. When they change the digit after the decimal point, you have a good, reliable piece of software,

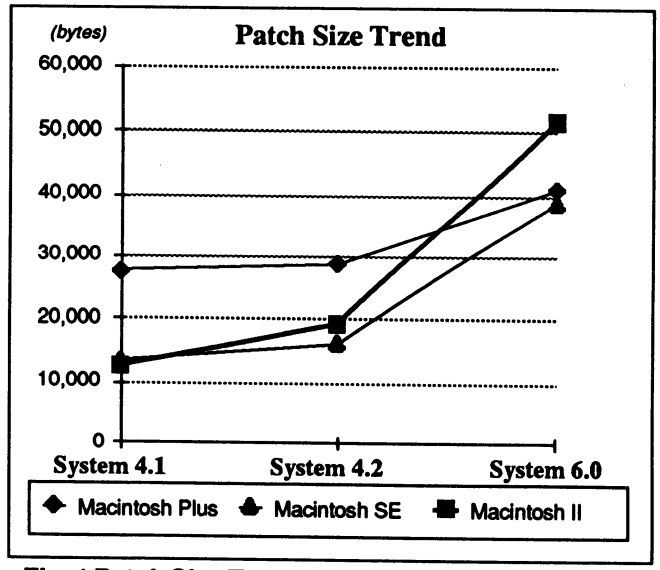


Fig. 1 Patch Size Trend in Bytes is going up, up, up!

because they were mostly fixing bugs, not adding them.

Note that the table offers the most optimistic viewpoint of compatibility; if the version was ever considered compatible, I list it as compatible, although usually the latest version has the most up-to-date bug fixes. Also, a 512Ke with a 1 Mb or more (third party or otherwise) can be treated just like a Plus; otherwise, taking a 512K memory configuration beyond System 3.3 is not such a great idea.

Not shown in the table is the great improvement in user-identifiable version information now included in all system software. The 'vers' resource, when properly used by any programmer, provides the user with a way to unambiguously identify the version of software or a document. More on that another time.

Passels of PC's

A year ago, I was writing the earlier article on my Macintosh Plus; thus, my testing emphasized the Plus, because it was easier much more accessible.

A full year later, I'm back to a Macintosh Plus again. However, this year I was able to test on Macintosh II and SE's owned by Palomar Software.

Of the Mac II's at the office, we have both the original ROM and the second revision (which provides 32-bit Slot Manager support). However, it should be noted that the 'PTCH' resources don't distinguish between the two, patching the same traps for each. Also, the location of all ROM-based traps were the same, lending credence to the speculation that the changes were limited in nature.

© The Definitive MacTutor, Vol. 4

Any Similarity to Real Traps...

As I've said before, Apple doesn't show me ROM listings, which, under trade secret law, allows me to write what little I know and publish it in *MacTutor!*

I used the same algorithm as in Part I, which compares the address of each trap to the standard unimplemented trap, $A89F. The sample program shown there is unchanged.

This gives me the availability of each trap by trap word ($A000-AFFF) with certainty, both for the traps that have names, and those that do not. But for the dispatched traps, I can't tell when (or which) new traps are added that share the same dispatch word.

Table 1: System software versions

Date	Official Name	System	Finder	MultiFinder	Compatible with 128	512	512Ke	Plus	SE	II
Jan 84		1.0	1.0		o					
May 84		1.1	1.1		o	o				
Apr 85		2.0	4.1†		o	o				
Jan 86		3.0	5.1		-	-	-	-		
Jun 86		3.2	5.3			o	o	o		
12 Jan 87		3.3	5.4			o	o	o		
15 Jan 87	1.0	4.0	5.4				o	o	o	
14 Apr 87	2.0	4.1	5.5				o	o	o	o
12 Oct 87	5.0	4.2	6.0	1.0				o	o	o
30 Apr 88	6.0	6.0	6.1	6.0				o	o	o

† Resource fork labels this "Finder 2.2"

Table 1: Which System for Which Machine?

As before, I've left off those traps for which I have discerned names but are not supported by Apple. Even more traps are defined but not named outside Apple—although it's interesting to note that System 6.0 seems to *remove* many of these traps from the trap table, marking them as unimplemented.

Apple does provide some clues as to why the changes were made, which were of great help.

Bigger and Better

Extensible software is one of my fetishes; my main complaints against the Mac OS and Toolbox design are in those areas where it is not extensible.

Thus, in hindsight, I can say that the RAM-based trap patches introduced by Apple about two years ago were a brillant move. One look at the size of today's patches and the functionality changes they provide will show how successful the concept has been.

As we saw in the last episode, trap patches have three purposes:

- Fix a bug;
- Extend existing capabilities
- Add a new trap.

In System 4.2 and 6.0, very few new traps were defined. Instead, the most radical difference since 4.1 comes with MultiFinder, which defines a few new traps and, more significantly, redefined many more.

As before, the 'PTCH' resources are used to apply machine-specific trap patches. New are the 'ptch' resources, which define patches common to two or more machine types. This reduces the size of the System file by removing duplicate patches, although the actual memory used may be higher.

In fact, the trend has been towards more and more memory allocated for patches. Table 2 lists the sizes of each patch resource, and the total size of the resources (approximately the total RAM required for the patches).

As shown in Figure 1, the Macintosh II has increased the most. It once had the most up-to-date ROM, but with most of the patches in the other machines, plus those required for Color QuickDraw, it now leads the way.

Unlike our last episode, Apple now approves allowing the user to install a stripped System disk with only those patches needed for their particular machine. (I guess the problem of building a MultiFinder master disk gave this an added urgency.)

If the user tries the stripped System on another model, an error alert is given at boot time and (s)he's told to go back to the drawing board.

MultiFinder

MultiFinder has a major effect on the run-time environment of an application, including the available traps.

As shown in Table 3, MultiFinder defines two new traps. _WaitNextEvent is a _GetNextEvent designed for non-preemptive multitasking, while _OSDispatch is currently used only to get at MultiFinder temporary memory management routines.

Even without MultiFinder, _WaitNextEvent is implemented in System 6.0. If you wondered why Apple advised you to check the availability of the two traps separately, wonder no more.

MultiFinder also patches a large number of traps to augment windowing, event, file control and memory-related functions. Changes also affect the Standard File Package and several Resource Manager calls, as shown in Table 4.

Other New Traps since 4.1

The new traps have been defined since the earlier episode are listed in Table 5. All but one of the traps are new with System 6.0.

One new trap, _LwrString, is a complement to the long-standing _UprString in the OS Utilities. It should be used only as a quick&dirty way to lower-case Roman text; a more portable call is to use the Script Manager transliteration routines.

All machines get a new manager in System 6.0, the Notification Manager. This provides a structured way for a background application to get a user's attention, as PrintMonitor does under MultiFinder.

The Notification Manager requires two new traps, _NMInstall and _NMRemove, which add and delete notification requests from a system-maintained queue. Everything you ever wanted to know about the Notification Manager is in *Macintosh Technical Note #184*.

The Notification Manager traps are stored 'ptch' #2.

Two traps that are not new to most programmers are _Debugger and _DebugStr. However, they are now provided by default by System 6.0, even if no debugger is installed.

I think this is great. Suppose you accidentally leave debugging code in your program and give it to an unsophisticated user. Your debugging calls become no-ops instead of ID=12 bombs: which do you think the user would prefer?

Finally, System 4.2 introduced one new trap for the Macintosh II only. The Pallette Manager was extended by _CopyPalette, for making a copy of a palette.

RAM-based traps

Although the Macintosh II included a new Sound Manager, this was not available across all machines. System 6.0 remedies this situation by providing the same RAM-based Sound Manager for all three machines. This provides bug fixes to the earlier version in the Mac II ROM.

The Sound Manager patches for all three machines are stored in 'ptch' #3.

Extended TextEdit, originally in RAM for the Mac Plus only, has been corrected and improved since System 4.2. Again, for all three machines, the manager is RAM-based, even though the Mac II has earlier versions in ROM.

These TextEdit patches are stored in 'ptch' #0.

Mac II Teething Pains

New babies have teething pains; the Macintosh II was no exceptions.

It appears that a large amount of new code was written for the Mac II ROM. Even for existing traps, some may have been recoded in 68020 for speed; other code was required to support many of the Toolbox extensions, such as auxillary window records.

We all know there were severe time constraints in getting it out the door, so it's not suprising there were problems, although I never noticed fatal problems associated with using my Mac II (except brain-damaged code incompatible with a 68020).

Both System 4.2 and 6.0 include a large number of traps patched only for the Macintosh II. These are listed in Table 8 and 9.

System 6.0 includes a much more robust and aggressive Palette Manager than was included in System 4.1, where it was a last-minute addition. However, the Palette Manager is not shown in the list of trap patches, because it has always been RAM-based, so the current testing methodology will not show any changes. However, changes to the two Color Manager calls _SaveEntries and _RestoreEntries are a tip-off to this color rewrite.

I should also note that a few of these routines may also have been patched to fix problems in the Macintosh Plus and SE. As noted, these routines were already patched in System 4.1 to provide new functionality (notably hierarchical menus), so I couldn't tell if they also had been changed by 4.2 or 6.0.

Other Bug Fixes

The remaining trap patches are shown in Table 10.

System 4.2 fixes the notorious early bug with _ADBReInit, while updating the _KeyTrans implementation.

System 6.0 improves the behavior of the standard polygon-drawing calls with large coordinates. Even if only a few points were defined, large enough coordinates would crash QuickDraw (without warning).

This was first pointed out to me by Jean-Paul Harmand at the Paris developer's conference in April. His program is shown in Example 1; it works fine with 6.0, but crashes with ID=25 on System 4.2.

There were quite a few places were traps were *unpatched* by one of the new releases. Since I haven't the foggiest idea of how to explain anything like that, I didn't list them, except for those traps patched in System 4.2 but not in 6.0.

Hidden Changes

I can't tell directly when a trap patch changes: if it was previously patched, all I know is that it remained patched, not whether the patch is the same.

In addition, for the dispatched traps, I can't tell which new traps are added that share the same dispatch word.

There are two areas that surely changed but don't show up on the list.

First, System 6.0 includes major changes to software related to internationalization. This includes the International Utilities and the Script Manager. A brief list of the new calls is given in Table 11. Except for the previously-mentioned _LwrString, all of these calls are part of a dispatched trap, either _Pack6 or _ScriptUtil, respectively; thus, there's no way of knowing they're there, except from the documentaton.

Secondly, _SysEnvirons has been updated for each machine and system version. Needless to say, this will always be a RAM-based call.

Printing Glue

In the earlier episode, I spent some time on the trap-based Printing Manager introduced in System 3.3.

Programmers who can verify (or count on) System 3.3 or later should use the traps, since this is the official, supported way from now on.

However, MPW programmers who still use glue may be in for a suprise. In MPW 2.0, the C and Pascal glue libraries first check for the trap to be implemented; if it is, they call it instead of performing the original Lisa Pascal glue functionality.

Acknowledgements

Since System 4.1, Apple has been including valuable release notes with each mailing of the new disks to developers.

The release notes only keep getting better; in the case of 6.0,

it even includes a list of all the managers, any changes and their current known bugs.

I'm sure I speak for all developers when I say the more timely information like this, the better. This information was extremely valuable, both in tracking down our own compatibility problems, and in preparing this article.

If you haven't figured it out already, you should grab all the release notes in your possession, shove them into a binder and file it next to *Inside Macintosh* and your tech notes in the programmer library.

Tech Support also provided me with an early draft of the Script Manager 2.0 documentation. I was about to sit down and write a two-part series on internationalizing software until I heard about 2.0, so I've held it up until I can fully grasp all the information. Look for it in a future issue.

Corrections

This all started when I was writing *Programming with Macintosh Programmer's Workshop*. If you want a list of all the traps up to System 4.1, see Appendix E, but please excuse some of the formatting problems.

In the prior episode, I said the Levco Prodigy (for the Mac Plus) patched certain unnamed traps. Duane Maxwell of Levco promised that he hadn't, but this information didn't make it into the earlier article in time for publication.

Table 2: Trap patch sizes

	System 4.1	System 4.2	System 6.0
Common patches			
'PTCH' #0	540	638	826
'ptch' #0			10,214
'ptch' #1 (Plus,SE)			4,762
'ptch' #2			3,474
'ptch' #3			3,866
Subtotal	540	638	23,142
Macintosh Plus			
'PTCH' #117	26,884	27,916	18,080
Total	27,424	28,554	41,222
Macintosh SE			
'PTCH' #630	12,958	15,622	15,712
Total	13,498	16,260	38,854
Macintosh II			
'PTCH' #376	12,004	18,724	33,136
Total	12,544	19,362	51,516

Table 3: New traps in MultiFinder

Name	Word	5.0	6.0	MultiFinder
_WaitNextEvent	A860		•	•
_OSDispatch	A88F			•

Table 4: Traps patched by MultiFinder 6.0

Name	Word
_Open	A000
_Close	A001
_GetVolInfo	A007
_ReallocHandle	A027
_SetTrapAddress	A047
_HNoPurge	A04A
_RDrvrInstall	A04F
_NewHandle	A122
_HFSDispatch	A260
_SetCursor	A851
_CalcVis	A909
_CalcVBehind	A90A
_ClipAbove	A90B
_PaintOne	A90C
_PaintBehind	A90D
_NewWindow	A913
_SelectWindow	A91F
_BringToFront	A920
_DragWindow	A925
_CloseWindow	A92D
_AddResMenu	A94D
_GetNextEvent	A970
_EventAvail	A971
_WaitMouseUp	A977
_ModalDialog	A991
_UpdateResFile	A999
_GetNamedResource	A9A1
_SizeRsrc	A9A5
_GetResAttrs	A9A6
_GetResInfo	A9A8
_SystemEvent	A9B2
_SystemClick	A9B3
_OpenDeskAcc	A9B6
_SysEdit	A9C2
_SysError	A9C9
_Pack3	A9EA

Table 5: Other New Traps

Manager	Trap	Word	Version	Machine
Script	_LwrString	A056	6.0	Plus, SE, II
Notification	_NMInstall	A05E	6.0	Plus, SE, II
Notification	_NMRemove	A05F	6.0	Plus, SE, II
Debugger	_Debugger	A9FF	6.0	Plus, SE, II
Palette	_CopyPalette	AAA1	4.2	II
DebugStr	_Debugger	ABFF	6.0	Plus, SE, II

Table 6: RAM-based Sound Manager

Name	Word
_SndDisposeChannel	A801

© The Definitive MacTutor, Vol. 4

_SndAddModifier	A802
_SndDoCommand	A803
_SndDoImmediate	A804
_SndPlay	A805
_SndControl	A806
_SndNewChannel	A807
_SysBeep	A9C8

ROM-based Sound Manager traps are available in the Macintosh II only. With System 6.0, RAM-based traps are defined for the Macintosh Plus, SE, and II. System 6.0 also patches the OS Utility _SysBeep is patched for the Macintosh Plus and Macintosh SE only.

Table 7: RAM-based TextEdit

Name	Word
_TESelView	A811
_TEAutoView	A813
_TEGetOffset	A83C
_TEDispatch	A83D
_TEStyleNew	A83E
_TEGetText	A9CB
_TEInit	A9CC
_TEDispose	A9CD
_TextBox	A9CE
_TESetText	A9CF
_TECalText	A9D0
_TESetSelect	A9D1
_TENew	A9D2
_TEUpdate	A9D3
_TEClick	A9D4
_TECopy	A9D5
_TECut	A9D6
_TEDelete	A9D7
_TEActivate	A9D8
_TEDeactivate	A9D9
_TEIdle	A9DA
_TEPaste	A9DB
_TEKey	A9DC
_TEScroll	A9DD
_TEInsert	A9DE
_TESetJust	A9DF

Table 8: Macintosh II System 4.2 patches

Name	Word
_Open	A000
_GetHandleSize	A025
_SwapMMUMode	A05D
_GetMaskTable	A836
_FixDiv	A84D
_NewRgn	A8D8
_MapRect	A8FA
_PaintOne	A90C
_PaintBehind	A90D
_NewWindow	A913
_CouldDialog	A979
_FreeDialog	A97A
_DiposDialog	A983
_CouldAlert	A989
_FreeAlert	A98A
_DetachResource	A992
_HandToHand	A9E1
_DisposPixPat	AA08
_DisposCCursor	AA26
_SetWinColor	AA41
_GetAuxWin	AA42
_NewCWindow	AA45
_GetNewCWindow	AA46
_DrawPicture†	A8F6
_InsertMenu†	A935
_CalcMenuSize†	A948
_CountMItems†	A950
_DelMenuItem†	A952

† Patched in System 4.1 and later for the Macintosh Plus and SE.

Table 9: Macintosh II System 6.0 patches

Name	Word
_DisposHandle	A023
_HSetState	A06A
_SetOSDefault	A083
_InitGraf	A86E
_InitWindows	A912
_GetKeys	A976
_NewPixPat	AA07
_SetDeskCPat	AA47
_SaveEntries	AA49
_RestoreEntries	AA4A
_EnableItem†	A939
_MenuSelect†	A93D
_MenuKey†	A93E

† Patched in System 4.1 and later for the Macintosh Plus and SE.

Table 10: Other patches

Name	Word	Plus	SE	II
_UnmountVol	A00E	6.0	4.2	4.2
_GetVol	A014	ram	4.2	4.2
_DisposPtr	A01F		4.2§	4.2§
_HLock	A029		4.2§	4.2§
_HUnlock	A02A		4.2§	4.2
_InitApplZone	A02C		4.2	6.0

_ADBReInit	A07B	-	4.2	4.2
_KeyTrans	A9C3	ram	4.2	4.2
_StdPoly	A8C5	6.0	6.0	ram
_PackBits	A8CF	6.0	4.2	
_StdBits	A8EB	6.0	6.0	
_StdTxMeas	A8ED	6.0	6.0	
_MoveWindow	A91B	6.0	6.0	ram
_InitResources	A995	6.0	6.0	6.0
_SystemTask	A9B4	6.0	6.0	6.0

Legend:

Notation	4.1	4.2	6.0
	ROM	ROM	ROM
ram	*RAM*	*RAM*	*RAM*
4.2	ROM	*RAM*	*RAM*
4.2§	ROM	*RAM*	ROM
6.0	ROM	ROM	*RAM*
-	(n.a.)	(n.a.)	(n.a.)

Table 11: New Script Manager calls

OS Utilities

UprText	Same as UprString
LwrText	Simplified change to lowercase

Script Manager

FindBlock	Break Roman text from native run
Form2Str	Display numeric format string
FormStr2X	Parse string to number using format
FormX2Str	Display number in string using format
GetFormatOrder	Arrange text for bi-directional format runs
InitDateCache	Initialize for String2Date and String2Time
LineBreak	Break line on word boundary
LongDate2Secs	Convert date with era to 64-bit integer
LongSecs2Date	Convert 64-bit integer to date with era
PortionText	Proportionate justification information
ReadLocation	Read latitude, longitude, GMT difference
Str2Form	Compile numeric format string
String2Date	Convert string to date
String2Time	Convert string to time
ToggleDate	Increase/decrease date field
ValidDate	Validate date
VisibleLength	Length of text excluding trailing white space
WriteLocation	Write latitude, longitude, GMT difference

International Utilities

IULDateString	Convert long date to string
IULTimeString	Convert long time to string

Source Code Listing: Quickdraw Crasher

```
; Sample program to crash QuickDraw with Polygon operation
; Original by Jean-Paul Hermand
; Transcribed to MPW by Joel West for MacTutor, 6/10/88
;
; Build using:
;   CreateMake PolyTest PolyTest.a
;   BuildProgram PolyTest
;
    Print   OFF
    Include 'Traps.a'
;   Include 'ToolEqu.a'
    Include 'QuickEqu.a'
;   Include 'SysEqu.a'
    Print   ON

QuickDraw   RECORD      ,DECREMENT
thePort     DS.L    1
white       DS.B    8
black       DS.B    8
gray        DS.B    8
ltGray      DS.B    8
dkGray      DS.B    8
arrow       DS.B    cursRec
screenBits  DS.B    bitmapRec
randSeed    DS.L    1
            ORG     -grafSize
            ENDR

    MAIN
PolyLen EQU $001E

    WITH QuickDraw
    PEA     thePort
    _InitGraf
    _InitFonts
    _InitWindows
    _InitMenus
    CLR.L   -(SP)
    _InitDialogs
    _TEInit
    _InitCursor
    MOVE.L  #PolyLen,D0
    _NewHandle

    MOVE.L  A0,A2
    MOVE.L  (A0),A0
    LEA     PolyValues,A1
    MOVE.W  #PolyLen/2,D0
@1  MOVE.W  (A1)+,(A0)+
    DBRA D0,@1

    MOVE.L  A2,-(SP)
    PEA     WhitePattern
    _FillPoly
    _ExitToShell

WhitePattern
    DC.L 0,0
PolyValues
    DC.W PolyLen
    DC.W $FD20,$FDC3,$165C,$07EE
    DC.W $165C,$07EE
    DC.W $1170,$02B4
    DC.W $02FF,$0021
    DC.W $FD20,$FDC3
    DC.W $165C,$07EE
    ENDMAIN

    END
```

Programmer's Tools

Prototyper, MacExpress, and FaceIt Compared

Doug Poland
Niceville, FL
MacTutor Vol. 4 No. 10

Three Programming Tools Compared

Introduction

The purpose of this article is to review several programming tools that make the job of providing your application with a Macintosh™ user-interface easier and quicker. While there are many different products of this nature, I can only reasonably discuss those products I have purchased and used.

This review deals with products that are procedural in implementation. An alternative to these types of tools is the use of an object-oriented programming (OOP) library as a starting point for your application. OOP is especially attractive on the Macintosh because of the strong visual metaphor utilized by its interface, and I would encourage you to obtain as much information as possible if you are intrigued by OOP.

Producing a Macintosh Application

Since you probably spend a great deal of time using a Macintosh, you are no doubt familiar with the minimum requirements of a good Macintosh application. Obviously, the user-interface is the characteristic that distinguishes Macintosh applications from other computer programs. In general, your application should support the user-interface (as described by Apple) to the extent reasonable, including the clipboard, file operations, desk accessories, printing and support the latest system software.

Implementing this type of interface is no small task. I would estimate that in a typical commercial application, as many as 5000-10000 lines of code are dedicated to supporting the features described above. Many programmers turn, at least in part, to previously written libraries of code or commercial application shells to accomplish this task.

This article discusses three products that address this need in varying degrees. The advantage of using these products is that they contain a great deal of "accumulated programming knowledge" related to implementing the Macintosh user interface and are excellent starting points for your application.

Languages

While the choice of a particular programming language is largely a function of which you are most comfortable with, some languages are better supported by programming tools than others and, in general, a compiled language is the best choice for a commercial application.

Pascal probably has the largest selection of programming tools, but C, Assembly and Modula-2 (to name a few) are popular languages for the Macintosh that are supported by application "shells". Fortran, an excellent engineering language, is not supported by many programming tools. However, as you will see, it is supported by at least one excellent product.

Another issue that might affect your choice of a language and a programming tool is the level of support that they provide for the manipulation of resources. All good Macintosh programmers make use of resources. The resource fork of an application should contain all of the internal text, pictures, menus and windows of your program. In this way, they are easily modified without re-compiling your code. I typically use ResEdit to create and edit resources and this program is available through APDA.

Application Uniqueness versus "Shell" Support

When attempting to select an programming tool, it is important to evaluate how much support you need. This decision is largely a matter of how unique your application is in terms of the "typical" Macintosh application and the amount of time and money you can afford to invest in "programming the interface". Generally speaking, the more a programming tool or shell does, the less flexibility you have in implementing a truly unique application. I will call this characteristic the "support -vs- uniqueness tradeoff". Regardless of which tool you decide to purchase, it should provide (for whatever level of support it is designed to supply) functional capabilities as well as the expected visual properties of a good interface to your application.

A Trio of Hypothetical Applications

If you compare the vast majority of Macintosh applications, they generally support the generation and editing of textual, pictorial or tabular data. While this is a gross simplification it will provide some framework for the discussion that follows. Let's consider three programmers (with only a minor knowledge of Macintosh programming):

Programmer A is working on a complex engineering simulation that already exists on a mainframe. This program processes text files as input and generates text files and graphics as output.

Programmer B is working on a new application that is, essentially, an "iconic" programming language for educational purposes.

Programmer C is working on a text adventure game that requires a few relatively simple dialogs but relies primarily on keyboard input in use.

The characteristics of these applications are admittedly abbreviated in order to facilitate the discussion of the programming tools I will describe in this article. However, I think that they are representative of the types of programs an individual is likely to write. Mainstream software (e.g., word processors) are

so complex and well-supported in the market place that I think they are beyond the scope of this article.

Three Programmer's "Scope-Out" Their Work...

At this point, the three programmers introduced earlier are prepared to describe the general interface features needed by their programs.

Since programmer A's application makes heavy use of text files, it would probably be appropriate for the application to generate the equivalent input files through dialogs and/or supply the program with Macintosh-style text editors for the user's convenience. Since the application also generates graphics, it must draw and print these pictures and ideally support the clipboard sufficiently to move both text and graphics to other applications for embellishment. It is desirable (from a timeliness perspective) to use as much of the existing code as possible without sacrificing the valuable characteristics of the Macintosh interface (namely the concept of modeless operation). Programmer A decides to implement the program in the following manner:

The application will have one text editor which contains the input file for the simulation algorithm and its contents will be generated by the user's interaction with a dialog. To execute the simulation, the user selects a menu command and, when complete, the application will present its results in another text editor and a graphics window. At that point, the user can alter any parameters significant to the simulation by selecting a dialog or editing the text in the input editor and then "re-running" the simulation. The user can copy/paste the text and graphics associated with the application for further use in other programs.

Programmer B's challenge is to support a very forgiving graphical programming environment. This application must not only display graphic information, but must understand the significance of the position and relationships of all the iconic symbols it supports. The application might support multiple "views" of its documents and allow the scrolling and splitting of document windows to allow the user to look at discontinuous portions of the same document. The program should support printing and have some special purpose dialogs for its various features. It should support the copying and pasting of components between documents and be able to save all of a document's contents to disk. Programmer B realizes that this application has many visual similarities with an object-oriented drawing program. This programmer decides to implement the program in the following manner:

The application will have a "tool" palette of programming symbols and a "drawing" window. By double-clicking on an object created with one of these tools, the user may view and change any data associated with it. Since the application's purpose is to teach "structured-programming" the selection/copy/paste of component(s) or module(s) is desired.

The vast majority of Programmer C's work will be involved with the processing of keyboard input and generating the appropriate responses. There are a few dialogs that allow the user to make selections among weapons or the like. Printing and clipboard support are not really necessary but saving the state of a game is important so the programmer must support file operations. Programmer C decides to implement the program in the following manner:

The application will make use of a single text editor for the entry of commands or questions and the response the adventure game computes. There are also a few menu items that are tied to dialogs for the selection of "weapons" or actions to take at any particular moment in the "adventure".

Introducing the Programming Tools

In the following sections, I will introduce the three programming tools that I have purchased and used. I must preface my discussion with the following statement: Products of this nature are constantly being improved and enhanced, and, therefore, you should always contact the publisher concerning the features of the latest version of a particular package prior to making a purchasing decision. The three programming tools that I will discuss in this article fall into two general "categories": source code generation and pre-compiled libraries. Prototyper™ is an application which generates resources and source code as output. MacExpress™ and FaceIt™ are pre-compiled libraries of "interface" code.

I will introduce these products in the order of their relative support for an application's interface.

Prototyper

Prototyper is an application that allows a programmer (or non-programmer) to "construct" an interface composed of menus, windows, dialogs and alerts in a "MacDraw™ - like" environment. The user can "link" buttons and menu items to windows, forming the essential structure of an application.

The characteristic that sets Prototyper apart from other applications is that it can generate not only the resources associated with the user-designed interface, but the Pascal source code (other languages will no doubt be supported in future versions) to implement this interface. This is the first commercial product, to my knowledge, that provides this capability. The significance of this is that the "interface designer" can create exactly the type of dialogs and windows that are appropriate for his or her application.

This product could be very useful when the interface for a potential application is being coordinated with non-programmers. It is easy to use, includes an example project, a thorough manual, and provides a good starting point for constructing the interface for your application.

MacExpress

MacExpress is a generic application that is best described as an application manager. It is composed of object files, interface files and a small file of "core" resources . It is available for several popular languages and compilers. The manual is top notch and includes several instructive flowcharts indicating the relationship between MacExpress' routines/data structures and your application. There are several excellent example programs on disk. MacExpress adheres closely to the Macintosh user-

interface guidelines, supports desk accessories, handles the main event loop of your program, provides support for menus, handles the user's interaction with windows, supports the use of icons in a manner similar to the Finder, and provides assistance in creating "dialog" views.

MacExpress, while being procedural in implementation, presents the programmer with a very "object-oriented" environment where "objects" such as windows and icons are sent messages. The main event loop of your application resides in MacExpress where "raw" events are processed and translated into messages (you may pre-process these raw events for your own purposes). These messages are then handled by your own routines and/or passed on to the MacExpress module for processing. Programmers familiar with the technique of passing the address of routines in a procedure call will feel very much at home using this package. MacExpress is compact and extremely fast in use.

In MacExpress, objects have standard routines associated with their functionality. These standard routines can be overridden and/or used in conjunction with your own routines. The essential idea is that, for instance, a window is sent a message ("the mouse went down in my content region") and your application then responds appropriately via a "MouseDownProcedure" or just calls the generic procedure to handle this event.

The most attractive feature, in my opinion, of MacExpress is the sophisticated level of simplification it affords the programmer in creating windows. Windows may contain multiple panels all allowing resizing, splitting, scaling, scrolling, and autoscrolling. In addition to this, all of the functionality of normal "document" windows can be given to dialog "views" and MacExpress similarly delivers "processed" events in the form of "messages" to these composite windows.

FaceIt

FaceIt is a series of stand-alone code resources that provide your application with an excellent user-interface. FaceIt accomplishes this without linking and can be used by programs created with several languages. The manual is technical but complete and provides a thorough discussion of FaceIt's capabilities and how to access them. There are several documented example programs that demonstrate how to access the features of the product. FaceIt is heavily resource oriented and, a file of "core" resources acts as a starting point for the resource fork of your application, in a manner similar to MacExpress. There is also an "interface" file that contains the declarative header needed to access FaceIt's data structures and a short routine to "jump" to FaceIt's code resources.

FaceIt is "one-step" beyond MacExpress in terms of the support it affords an application. The price for this support is a reduced ability to generate very unique applications. However, a program utilizing FaceIt immediately has access to many features which are extremely difficult to implement from "scratch". FaceIt provides your application with text editors, a scrollable graphics window (with reduced view window), spreadsheet-type windows, complete printing support, high-level support for modal dialogs and alerts, support for background processing under MultiFinder™, support for DAs and the clipboard, and various memory and resource manipulation utilities.

In utilizing FaceIt, your program will implement a main event loop that handles program specific "commands" while FaceIt contains its own main event loop that directly processes "raw" events. At the beginning of your own program's "main event loop", your program passes control to FaceIt, and hence the user. FaceIt intercepts events; handling "generic" events (like opening a text file into one of the text editors) and passes control to your program's main event loop when one of your "program-specific" commands is requested.

Your program will access all of FaceIt's high-level interface utilities through a subroutine or procedure call. There are six (long integer) arguments involved in each call to FaceIt. These six arguments are single or multiple "macro commands" with any obligatory parameters that represent some unique functionality of FaceIt. Programmers new to the Macintosh will probably require little knowledge of the Macintosh ROM with the exception of QuickDraw™ (for generating lots of neat graphics).

Your program can create and/or access information in any of the windows, and it is very easy to implement complex dialogs with FaceIt. FaceIt handles the user's interaction with a dialog and provides very simple techniques for implementing scrollable lists, pop-up menus, and filter/activation-deactivation instructions, etc. FaceIt essentially reduces the task of implementing dialogs to that of simple string manipulation.

Back to Our Programmers...

Based upon the above discussions concerning the features and capabilities of each programming tool considered, I will now suggest which programming tool (among those covered in this article) would be appropriate for each programmer.

Since programmer A is involved in the generation of a complex application that already exists on another computer, this programmer is primary concerned with the amount of time and complexity that would be involved in adding an appropriate interface to the program. The best choice for this application is FaceIt. This package automatically provides text editors, a graphics window and "spreadsheet-like" windows all of which have "display options" that are user-configurable.

Because of the amount of interactive graphical support required by programmer B's project, FaceIt would not be a reasonable choice. The need for splitting and multiple views of the same document suggests that this programmer utilize MacExpress. This package would provide the most flexible "window environment" of any of the products discussed. Programmer B must write the code associated with updating the contents of the windows, printing, any required text-editing, file operations and clipboard support.

Although the application-specific complexity is by no means negligible for programmer C's project, the interface is quite simple. It would probably be best for this programmer to write this program from scratch while utilizing Prototyper to initially formulate the code and resources necessary for the menus, windows, and dialogs. There are several sources for text

editor algorithms and these packages, in combination with Prototyper, would probably be sufficient for programmer C's purposes.

Relative Strengths and Weaknesses of the Programming Tools

The products that I discuss in this article are, in my opinion, ideal for certain programming projects and not so ideal for others. The lack of certain features in a particular product does not necessarily represent a weakness or short-coming but might indicate a different philosophy concerning the proper "role" or level of functionality that these products should provide. With these comments in mind, I have compiled a brief summary of my opinions of each of the programming tools.

Prototyper

While Prototyper is not an application shell, it does have many features that make it ideal for the programmer new to the implementation of Macintosh dialogs, windows, alerts and menus. It is an excellent tool for the coordination of an interface "design" with non-programmers. Prototyper could be used to form the initial structure of your application's interface, be it large or small. Prototyper is also a very educational product with regards to the use of the toolbox in creating an interface.

MacExpress

MacExpress is a true generic application manager. It has been carefully structured to support the general concepts of an object-oriented program. It is small and very fast, and produces applications with an excellent interface. While you must write the code for printing, picture/text management, supporting the clipboard, file operations, and the integration of dialogs into your code, it is probably the most extensive shell that still provides the programmer with near complete freedom in the design of an application.

FaceIt

FaceIt is a series of stand-alone code resources that implement a full-fledged application. While the programmer is limited in terms of the level of unusual or highly unique features his or her application may have, FaceIt provides complete support of printing, the clipboard, desk accessories and a very high level of support for background processing, picture/text handling and dialog management. FaceIt is also one of the few application managers that supports Fortran and Basic.

Example Program

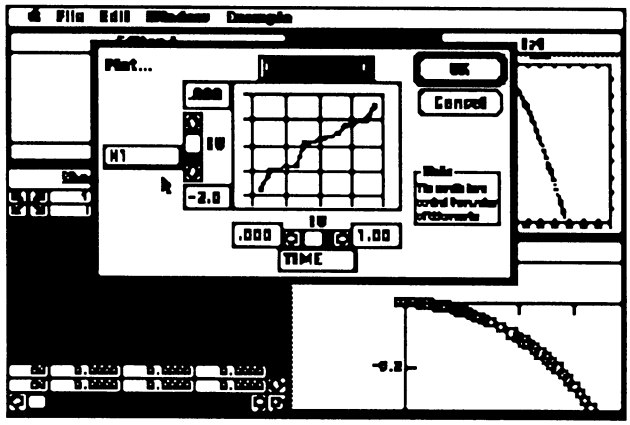
Since the purpose of this article is to describe programming tools and what they do for the programmer , I would be remiss if I did not include an example program.

The purpose of a programming tool is to assist the programmer in generating a good application by reducing the amount of time and the amount of code that he or she must write. As an example of what a well-designed programming tool can do to assist you in your efforts, I have included a listing and screen "snap-shot" of an example plotting application that utilizes FaceIt. This program makes use of FaceIt's "spreadsheet" and graphics windows. The example program (in Fortran) demonstrates the integration of FaceIt into your application.

```
      program examp
      implicit none
      .
      …"type" declarations and initializations for program…
      .
      !Include FaceIt declarations…
      !include StorMF.inc

     !Load frequently used (but un-liked) subroutines into memory
      load JumpMF
      load toolbx

     !Inform FaceIt of application's resource file (if separate)
      name = 'examp.Rsrc'

      !Initialize FaceIt
     !   -> enable auto-opening of TEXT and PICT files from Finder
     !   -> set aside 50 Kbytes of space for stack expansion (for
pictures)
      call FaceIt(1,0,-1,50,0,2)

      !Enter your program's main event loop
      do
         !pass control to FaceIt (and hence the user)
         call FaceIt(0,0,0,0,0,0)

         !process a program specific command…
         select case(MAC)
            case('About')
               !the user chose the "About Program…" menu item
               !open the "About" alert  (resource ID 1099)
               call FaceIt(0,OpnAlt,1099,0,0,0)

            case('Open')
               !the user chose the "Open" menu item for the
spreadsheet window
               !present the user with the standard file dialog
               !restricting the files displayed to those of type
"TEXT"
               MAC='TEXT'
               call FaceIt(0,StdOpn,0,0,0,0)
               if (name<>'Cancel') then
                  !if the user didn't press the "Cancel" button
then…
                  .
                  …read disk file
                  …     assign numbers to the data array
                  …     assign column names to STR# resource 1001
using
                  …     FaceIt's SetStr utility…
                  .
               endif
               !display the array in the spreadsheet window…
               !     ->give FaceIt a pointer to the array to be
displayed
              !     ->display the array in the spreadsheet window
               arrayptr(1) = toolbx(PTR,dat)
               call FaceIt(1,SetSh1,500,10,0,5)

            case('Plot…')
               !the user chose the "Plot…" menu item

             !transfer the information for this dialog to FaceIt
               !    ->transfer dialog item list status
               do (i=1,16)
                  dialog(i)=plotdialog(i)
               repeat
               !    ->use list of column names (STR# 1001) in pop
menus…
```

```
            listID(1)=1001
            listID(2)=1001
              !     ->transfer textual information (text and
numbers) to FaceIt
              !       through the "name" variable (used as an
internal file)
                                                  write(name,30)
title,maxY,divY,minY,minX,divX,maxX

            !open the dialog (resource ID 2000) for manipula-
tion by the user
            call FaceIt(0,OpnDlg,2000,0,0,0)

            !after the dialog has been dismissed, process it…
            if (dialog(1)=1) then
               !if the user pressed the "OK" button then…

              !transfer the (changed) information from FaceIt
to the
               !program…
               !transfer dialog item status back to program
               do (i=3,16)
                  plotdialog(i)=dialog(i)
               repeat
                !transfer textual information (text and num-
bers) back to
                !program
                                                  read(name,31)
title,maxY,divY,minY,minX,divX,maxX
               !based upon the values of this info, prepare to
plot

                  …set a few variables based upon info from
dialog…

               !begin recording a QuickDraw picture (the plot)
                call FaceIt(0,NewPic,0,0,0,0)

                …draw the plot here…
                .     …draw the axes…
                .     …plot the points…

              !end the recording of the (plot) and display it…
                call FaceIt(0,RetCtl,0,0,0,0)
            endif

         case default
      end select
   repeat
30 format(A32,F20.6,I4,F20.6,F20.6,I4,F20.6)
31 format(A32,F20.0,I4,F20.0,F20.0,I4,F20.0)
   end
```

The following screen "snap-shot" captures the example program after the user has loaded some data into the "spreadsheet" window, produced one plot and is preparing to generate another.



What might not be immediately obvious is the number of features that this very small application has as a result of utilizing FaceIt. The user has access to "spreadsheet", text editor and graphics windows with the expected scrolling, selection and editing characteristics. The application offers complete support for printing and the clipboard in all of the windows. This application also sports a functional and intuitive plot dialog that I created with ResEdit. This dialog is displayed with a single call to FaceIt and features editable text items, pop-up menus and scroll bar controls (all of which are supported by FaceIt).

This small application is an excellent example of the value of programming tools or shells. These products have unique features specifically designed to reduce the burden of generating an application's user-interface, permitting the programmer to concentrate on the unique characteristics of his or her program and not become overwhelmed in the task of "programming the interface".

Additional Programming Considerations

After you have completed your application using any of these products, there are a few considerations you should take a look at concerning the use of the programs. I am referring specifically to disk space requirements and operating in a shared-application environment.

Ideally, your program should take up as little disk storage as is reasonable and operate smoothly under MultiFinder. While Prototyper and MacExpress (V1.14, new version appears to have utilities for working with MultiFinder) do not address these issues, FaceIt has some unique features in these areas.

FaceIt is really a "stand-alone" product. There are essentially three "places" that FaceIt can be located. It can be placed in the System Folder where it will be available to any program that uses it (without linking), it can be added to your program file to create a completely stand-alone application, or it can be added to the "System", where it is available to all programs that use it and only one copy is loaded into memory under MultiFinder. Either of these techniques can be used to reduce the disk space and memory requirements of your product (which might consist of multiple applications).

Making the Decision on a Programming Tool

Any recommendations that I make to you should be considered in light of my own programming experience. I am, for the most part, involved in large, technical programming projects. My programming time is spent in the details of implementing "engineering" concepts in code and, consequently, I do not have a great deal of time to devote to "programming the interface". In light of this information, here are my recommendations for the use of these programming tools:

If you are receiving interface input for your application from non-programmers and require several unique interface features but not a substantial amount of application support (you are writing the code for the clipboard, printing, window updating, etc.), or you just want to know how the Mac interface in dialogs

and menus is implemented, get Prototyper. Prototyper is easy to use and has several truly unique features that could make it very valuable to many programmers. Prototyper supports the following compilers: LightSpeed™ Pascal, Turbo™ Pascal, TML™ Pascal and MPW™ Pascal. Future versions of this product will probably support the C language. [$125.00, SmethersBarnes, P.O. Box 639, Portland OR, 97207, (503)-274-2800].

If your are developing a unique application in Pascal, C or Assembler and need speed, excellent window and application management, you are comfortable with writing the code for printing, file operations, clipboard and dialog support, then I would suggest you use MacExpress. This package is probably the best among those discussed in this article for creating a truly "unique" application. MacExpress supports the following compilers: MPW™ Pascal, MPW™ C, Consulair™ Mac C, TML™ Pascal, Aztec™ C, Lightspeed™ C, Lightspeed™ Pascal and various "Assemblers". [$195.00, Alsoft™ Inc, P.O. Box 927, Spring TX, 77383, (713)-353-4090].

If you are transferring an existing technical application (even in Fortran) or creating a new one and the amount of time and/or money required to "program the interface" is a critical issue, I would suggest using the programming package FaceIt. This is a particularly good choice if you find reading Inside Macintosh difficult. Integrating FaceIt into your application requires few changes to existing code and minimizes "toolbox" calls. The price that you pay for this level of support is an inability to implement any unique or unusual interface features. This package is certainly the "quickest" way to add a Macintosh interface to a program. FaceIt supports the following compilers: ZBasic™, LightSpeed™ C, MPW™ C, Language Systems™ MPW Fortran, MacFortran™, Microsoft™ Fortran, SemperSoft™ MPW Modula-2, Lightspeed Pascal, MPW™ Pascal, TML™ Pascal and Turbo™ Pascal. [$50.00 (single compiler), $100.00 (10 compilers), FaceWare, 1310 N. Broadway, Urbana IL, 61801, (217)-328-5842].

Closing Remarks

I hope that this short article proves to be helpful to you in your evaluation of a programming tool. As I stated earlier, this article is a brief review of only three packages that are currently available. I encourage you to "shop-around" and obtain as much information as you can. Since a programming tool may, to a large extent, determine the quality of your application's interface, it is vital (to your application's success!) that you use the proper tool. Good Luck!

Programmer's Forum

OS/2 Presentation Manager

Dan Weston
Portland, OR
MacTutor Vol. 4 No. 12

The Good, the Bad, and the Ugly: OS/2 Presentation Manager for Mac Programmers

by Dan Weston
Nerdworks, Portland, Oregon

The Presentation Manager (PM) is Microsoft's and IBM's answer to the Macintosh user interface toolbox. Due out in late 1988 with OS/2 version 1.1, PM provides a set of new operating system functions for application programmers that is somewhat similar to the functions supplied in the Macintosh ROM. OS/2 programmers can access these functions to create windows and menus, use the mouse, and do all the things that Mac programmers have been doing for over four years now. PM is directly descended from Microsoft Windows, so programmers familiar with Windows will have little trouble making the switch to PM, although their Windows application source code must be substantially modified to work with PM.

This article will attempt to explain the main features of PM in terms that Macintosh programmers should be familiar with. I know that many people in the Mac community look at OS/2 with deep suspicion, but even if you feel that way, read on, and you may find that there are things to appreciate in PM.

Events and Messages

The foundation of any Macintosh program is the event loop. The Macintosh operating system watches the outside world, including the mouse, keyboard, and disk drives, and signals the application whenever an event occurs. Macintosh programs are event driven.

In a PM program, messages take the place of events. When mouse and keyboard events take place, PM tells the application by sending messages to it. This is conceptually very similar to the Macintosh event loop, but PM has a much richer set of messages and has made the message architecture more general and extensible so that it is very easy for applications to create new message types.

Many of the messages have direct anologs in Mac event types, such as mouse button down and up messages (although there can be up to three mouse buttons), activation and update messages, and keyboard messages. Macintosh programmers familiar with event-driven application architecture will not find it too hard to make the adjustment to a message driven system.

Windows

The main components of a PM program are windows. The system provides many predefined window types, including frame windows, push buttons, radio buttons, menus, and title bars. The predefined window types are know as "window classes". Each window class defines an object that enclose a set of functionality that is available to the programmer simply by sending messages to a window with that type.

The messages in a PM system are actually sent to an application's windows rather than the application itself. Each window has an associated window procedure that receives messages from the system and other windows. For example, when the user presses the mouse button, the system sends a mouse button down message to the window that is underneath the mouse at the time of the event. The window then processes the mouse down message as it sees fit. Different types of windows respond differently.

A standard PM window, with title bar, size border, and scroll bars, is actually made up of several different predefined window types. The most basic is the frame window. On top of the frame window sits the other windows, such as the title bar window and the scroll bar windows, that gives the window its distinctive look. The frame window "owns" the other windows, called control windows, that sit on top of it. The ownership relationship is important since it allows the control windows to communicate with the frame window.

For example, the title bar window class responds to a mouse down event by tracking the mouse and allowing the user to drag an outline of the frame window around the screen to position it, just like on the Mac. The way this works is that the title bar window handles all the tracking and the display of the window outline. When the mouse up message comes to the title bar window, the title bar window sends a message to the frame window telling it the new position on the screen. The frame window then sends messages to all the windows that it owns telling them to redraw themselves at the new position.

Typically, a programmer writing a PM program creates at least one new window class to make a program. In the terminology of PM, this is the "client window". A client window corresponds to the content area of a Macintosh window. The client window is also owned by the frame window. Its main responsibility is to display data for the user. So while frame windows and their constituent control windows behave the same in most programs, it is the client window which gives each application its visual personality.

The operating system takes care of routing messages to the appropriate window. An application programmer never sees events that are handled by other windows. For example, when a normal Mac program gets a mouse down event, it must call FindWindow to determine what part of the window received the click and branch accordingly. In a PM program, the application programmer never sees a mouse down in the title bar or menus because those events are handled by the control windows at those locations. The application programmer only handles messages

© The Definitive MacTutor, Vol. 4

directed to those windows he or she creates, such as the client window mentioned above.

The Window Procedure

Each window has an associated window procedure that processes all messages for that window. The syntax for a window procedure is consistent for all windows, so all messages are essentially function calls to the window procedure of the window receiving the message. A very simple window procedure is shown below.

```
MRESULT MyWindowProc(hwnd,msg,mp1,mp2)
HWND hwnd;
USHORT msg;
MPARAM mp1;
MPARAM mp2;
{
    HPS hps;
    RECTL   rect;

    switch (msg){
    case WM_PAINT:
        hps = WinBeginPaint(hwnd,NULL,&rect);
        WinFillRect(hps,&rect,CLR_WHITE);
        WinEndPaint(hps);
        return (MRESULT)0L;
    default:
        break;
    }
    return WinDefWindowProc(hwnd,msg,mp1,mp2);
}
```

The window procedure is normally a long switch statement that decodes the message and responds appropriately. A key feature is that the window procedure calls **WinDefWindowProc**, for all messages that it does not handle it. WinDefWindowProc is the default window procedure supplied by the system. The default window procedure implements the default window behavior. Individual window classes handle only those window messages that they want to modify, passing the rest on to the system. This is a marvelous model for programmers because they can utilize all the default behavior without having to know how it works.

The arguments to a window procedure always follow the syntax given above, but the contents change depending on the particular message. The arguments are summarized below:

hwnd: a window handle to the window receiving the message. (Note: handle has a different meaning in PM than in the Mac. A PM handle is more like a token that the system knows how to use to find the window data structure.)

msg: a 16 bit integer that identifies the message type.

mp1: a 32 bit value whose interpretation depends on the message type.

mp2: a 32 bit value whose interpretation depends on the message type.

Creating Window Subclasses

Application programmers normally create their own window class and supply a window procedure for that class. They must register the class and its associated window procedure with the system. Once registered, the window class can be used to create multiple instances of windows of that class, all of which use the same window procedure. The window procedure created by an application for a new window class handles selected messages and passes the rest on to **WinDefWindowProc** to get the default behavior. This makes creating new classes very simple.

Another way to create a new window class is to subclass an existing class, using its window procedure as the default instead of **WinDefWinProc.** For example, suppose you wanted a new title bar type that drew its contents in italics instead or regular text. You could create a new class based on the standard title bar window class by subclassing the title bar class.

When you create a window subclass in PM, you get the address of the window procedure for the original, or parent, window class. You then construct a window procedure for the subclass that calls the parent's window procedure instead of **WinDefWinProc**. That way, all messages that your window procedure doesn't intercept are processed as they normally would be by the parent class. Going back to the example of a title bar that used italics, it would only be necessary for the subclass's window procedure to respond to the WM_PAINT message by drawing the title bar as it saw fit. All other messages could be passed to the original title bar window procedure. You don't have to reinvent the title bar, just change those parts you want to change.

Subclassing is a very elegant and simple way to extend or modify the behavior of the system. PM makes it easy to do.

Window Updating

Window drawing in the Mac environment is triggered by the update event. The Window Manager keeps track of the update region and generates an update event whenever a portion of the window needs to be redrawn. Smart Mac programmers utilize this mechanism by isolating all window drawing within the code that handles update events. In order to force a portion of the window to be redrawn, these programmers add that portion of the window to the update region and rely on the update event mechanism to call their drawing code.

The PM drawing architecture is very similar. All windows have an update region. When the update region is not empty, the system sends a WM_PAINT message to the window. All windows should respond to a paint message by drawing themselves.

Your client window procedure should respond to a paint message by drawing a representation of the current state of the data associated with the window. For a word processor, you would draw the text and graphics of the document.

Just like the Macintosh, PM clips all drawing in a window during the paint message so portions of the window which don't need to be redrawn are not touched by graphics operations.

Mac programmers familiar with the update event mechanism on the Macintosh will have no problem adjusting to the PM method of painting windows.

Menus

Menus in PM are a special predefined window class that sit

on top of a frame window just like the title bar and scroll bar. Each frame window can have its own set of menus. There is no system wide menu bar.

Since menus are actually windows, they know how to handle all sorts of messages, including user interaction with the mouse and keyboard. The menu window typically handles a mouse click by tracking the mouse until the user makes a final selection from the menu items. At that point, the menu window sends a command message to the frame window telling it what menu item was chosen. The frame window normally passes the command message on to the client window, where your window procedure receives the command message and reacts accordingly.

Thus, you are not responsible for dealing with menus until an item is actually chosen. The menu window, because it is an autonomous window object with built in functionality, is able to operate independently. You, as the programmer, are responsible for defining the contents of the menus and placing them in the frame window, but once they are in place they function on their own, notifying you only when an interesting event takes place.

PM menus are normally defined as resources, much like the Mac. The standard development tools from Microsoft include a resource compiler that processes resource definition files and creates resources which are then joined with the code to form the executable files. A menu resource definition includes the title of the menu, the text of each item, and the command number associated with each item. The command number for the item is sent as part of the command message when the item is selected.

One nice feature of PM's resource compiler is that it will read the same C header files as the C compiler. Thus, you can include the same .h file in your source code and your resource definition file to define the constants that correspond to the command numbers for the menu items.

Graphics

The Macintosh has Quickdraw. PM has the Graphics Programming Interface (GPI). GPI is based on an IBM mainframe graphics system. It is big and complicated and powerful, offering many transformation functions and a store and playback mode similar to QuickDraw pictures.

Where Quickdraw has the GrafPort, GPI has the presentation space and device context. The device context is probably closest, conceptually, to the GrafPort, in that it is where the graphics functions are translated into the operations that actually twiddle the bits on the device, be it a screen or a printer. But GPI has a higher level gateway, called the presentation space, that routes graphics commands to the device context. The presentation space takes care of some of the advanced transformation operations and the stored graphic playback operations.

GPI is harder to use than Quickdraw, but it is more powerful. Mac programmers will have to start from scratch with GPI since there is little overlap between the two systems.

Multitasking

The current Macintosh model of multitasking implemented in MultiFinder is based on a non-preemptive model. The fundamental unit of multitasking in the Mac is an application. Basically, a application is allowed to run in the background whenever the active application is waiting for an event. This is a good way to provide multitasking and still be compatible with old software, and many programs are already taking advantage of this feature.

In PM, the basic unit of multitasking is a thread. A process (application) can be made up of one or more threads. All threads in the system share the processor by time-slicing, so they appear to execute concurrently. A thread is a very efficient unit of processing, since it shares global data and most of the machine state with its parent process. The thread has only its own stack and register set, so the time required to context switch between threads of the same process is very short.

The idea in PM is that you spin off a new thread whenever you need to do something that will take you away from the main user interface thread for more than 1/10th of a second. For example, if the user selects a menu item for a lengthy calculation or sorting operation, the application should create a new thread to do the operation. The user interface thread will continue to execute concurrently with the calculation thread so the user is not cut off from selecting other menu items or switching to another application while the computation progresses.

In essence this means that you should never have to put up a watch cursor.

OS/2 provides semaphores and other classic operating system mechanisms to help synchronize and protect shared resources in a multitasking environment. For example, in the previous example of a lengthy calculation, you would want to protect the data with a semaphore so that the user could not change the data as it was being used in a calculation.

I think that threads may be the neatest thing about OS/2. I wrote a small sample program on both the Macintosh and OS/2. The program continuously calculated the roots of a chaotic quadratic equation and displayed the results as a graph. On the Mac, I used null events to run in the background under MultiFinder. In PM, I created a calculation thread that sent a message to the graphing window each time it calculated a new point on the curve. The graphing window then plotted the point and waited for the next message. The PM version of the program ran quite a bit faster than the Mac version (Mac II versus Compaq 386/20). I don't want to make this out to be some sort of benchmark, but I think that the ability to do real multitasking with threads will open up many new possibilities for programs.

Of course, multitasking also creates many new problems for programmers who are not used to protecting data and synchronizing multiple threads, but OS/2 provides the tools needed to solve these problems.

Interprocess Communication

PM provides a clipboard mechanism that is a superset of the Macintosh clipboard model. One nice feature of the PM clipboard is that an application can send several formats to the clipboard without actually rendering the data. The application only has to render the data in a particular format if another application makes a request for that format.

Beyond the clipboard, PM provides several other inter-

process communication (IPC) mechanisms, including shared memory, pipes, and queues. In addition to these traditional methods, PM applications can communicate by sending messages back and forth between their respective windows. All in all, PM provides a richer set of IPC tools than the Macintosh, and because these capabilities are present from the start, it seems probable that many application developers will take advantage of them.

Development Tools

Microsoft strongly suggests that you write your PM programs in C. The standard development environment is their C 5.10 compiler and linker, combined with the editor of your choice. Microsoft also provides an editor, which is programmable and extensible and totally configurable and takes about two weeks to learn how to use. Once you get over the learning curve, the tools aren't so bad, but friendly they aren't. It made me appreciate how good Mac developers have it with Lightspeed and MPW.

The best part of the development environment, however, is CodeView, Microsoft's source level debugger. CodeView will let you step through your code in source mode, mixed mode, or assembly language mode. It lets you look at variables and set break points and break conditions. It is truly a great tool. I spent many hours just watching messages flow into my window procedures while learning PM.

The other great part of the Microsoft development package is QuickHelp, which is an on-line reference to the system that can be integrated with the editor. It works like this: you type a function name or structure name or message name in your code and want to know more about it. Position the cursor somewhere in the name and press Alt-Q; you immediately get full information on the topic. Great tool!

I imagine that third party developers will eventually provide alternative development environments. The pre-release versions of Microsoft's software development kit (SDK) cost $3000, although that price included incremental upgrades over a year-long period. Clearly, not too many people are willing to pay that much, but I am sure that final retail product will be cheaper by an order of magnitude.

The Good

I like the messaging architecture of PM. It is actually fun to program using window procedures and messages. This is the most painless object-oriented programming system I have encountered. It was easier for me to learn than MacApp. There are many similarities between MacApp, especially version 2.0, and PM. MacApp 2.0's use of nested views is very similar to PM's nested windows. MacApp, however, is a true object oriented environment while PM is only partly object oriented. I think that the design of PM benefitted from Microsoft's experience with the Mac and two versions of Microsoft Windows.

Individual menu bars within windows are great. [See the November issue of MacTutor. -ed]

Even though the Z86 has a stupid 64K limit on segment size, the built-in memory management and protection between applications is nice.

The graphics are very powerful; in some ways reminding me of PostScript. (see also the graphics entry in The Bad).

The multitasking model is more advanced than the Macintosh, but that is understandable since Microsoft had the opportunity to design it in from the bottom up while the Mac folks had to graft it on top of an existing single-tasking OS.

CodeView and Quickhelp are incredible aids to learning PM. They beat anything I have ever seen on the Mac, although I haven't seen Lightspeed C's new debugger yet. If I had had the equivalent of CodeView when I started on the Mac I could have learned the Mac in half the time.

The Bad

The message-based, object oriented nature of PM has a downside. Much of the functionality of the system is hidden from the programmer. You inherit an enormous amount of default behavior. This is great until you decide that you want to change the default behavior. Then you need to know what to change. PM provides a clean way to change the system behavior by subclassing the predefined window classes, but the problem is that it is not always clear just which messages to intercept in order to change a behavior. With other object-oriented systems, such as MacApp, you get the source code for the system objects so you know exactly how they work and can use that code as the basis for your own subclasses of those objects. Microsoft does not provide source code for its standard window classes, so you must rely on the documentation to know how they work. This is a real problem if you want to do something that the documentation team didn't think of.

The graphics functional interface is too complicated. It is hard to do simple things. It is not much harder to do really hard things. Some people will love this graphics library, others will hate it. It lacks the elegance of Quickdraw, plus there is no Palette Manager to arbitrate colors among windows sharing the screen.

The file system is compatible with DOS, so you are limited to xxxxxxxx.xxx file names. Ugh!

The Ugly

The Macintosh screen is beautiful. I think all Mac programmers have a deep visual response to the screen. The Mac has lots of little touches that make it easy to work at all day: drop shadows, proportional fonts, square pixels. The PM screen is pretty ugly. I think the difference is the lack of graphic artists on the design team. I think the visual look of the screen was designed by programmers who were thinking about how difficult it would be (and looking over their shoulders at the schedule IBM is dictating for them) to implement the small touches that aren't there such as drop shadows. Of course square pixels are out because of the pc hardware, so you can't really blame the programmer's for that one.

I get the impression that there is no Steve Jobs at Microsoft, telling folks that just because something is hard to do is no reason not to do it. Many of the little compromises made in the interface in the name of simplifying the coding have hurt the PM look-and-feel. PM just doesn't feel as good as the Mac.

Another ugly point is that PM takes at least 2.5 megs of RAM; 3-4 megs is more realistic if you want to run several

programs at once. Also, although PM will run on a 286 based machine, you really need a 386 to make it responsive.

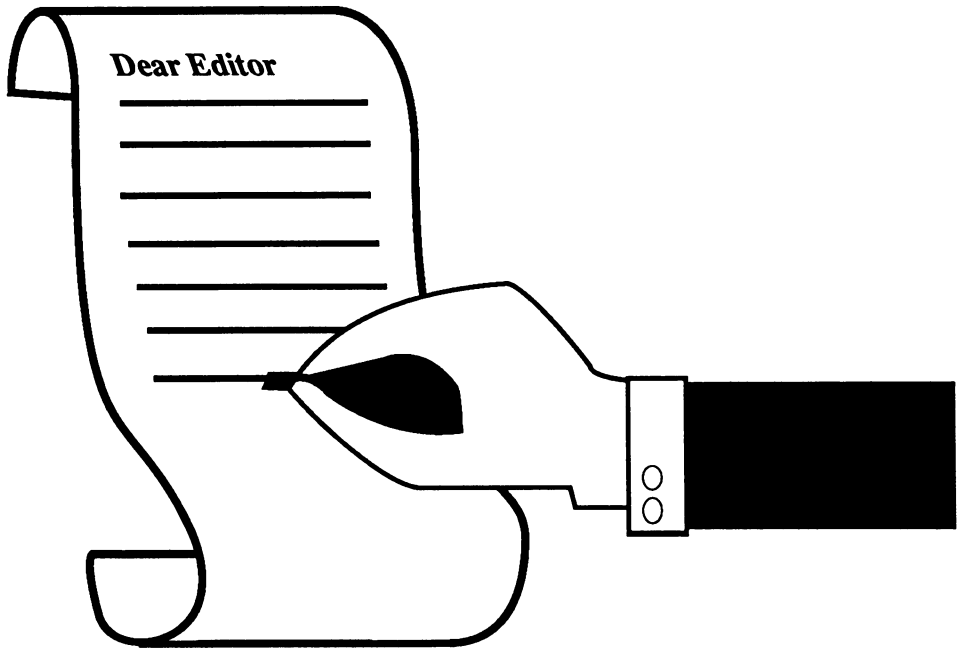
Conclusions

PM is fun to program. The messaging architecture is well-designed and allows you to put together very powerful programs pretty quickly. One of the main people behind PM is Neil Konzen; a legendary hacker and the author of the original Microsoft Multiplan for the Mac; so a lot of good experience and thought has gone into it.

PM is big and complicated. Expect a learning curve at least as steep as the Mac, although your Macintosh experience puts you way ahead, conceptually, of the old line DOS programmer trying to make a transition. The windowing environment is every bit as rich as the Mac with the added complication of multi-tasking. Be prepared to have your head hurt sometimes.

As to how you can program in both environments, there seems to be some emerging products based on a common library idea. Basically, you write your program, or at least part of it, by calling functions from a library that can be expressed either in Mac ROM calls or PM functions. The library is a higher level expression of the common features of both operating systems. Such products exist for cross development between the Mac and Microsoft Windows, so I have no doubt that they will be developed for PM as well. But these libraries cannot hope to cover some of the more fundamental architectural differences between the two systems, such as the multitasking models. I believe that portions of your programs will have to be hand crafted for each operating system.

The bottom line is this,PM is a rich, full-featured, operating system that will allow programmers to write applications that are as powerful and easy to use as anything on the Macintosh. Do not count it out. I think it will be a year or more before it really catches on because of the expensive hardware requirements and the lack of significant applications, but eventually it will be a big factor in the microcomputer industry.



Letters Column

© The Definitive MacTutor, Vol. 4

Letters & Editorials

David Smith
Editor & Publisher

Volume 4 Number 1

Tool Design the MultiFinder Way
An Editorial

Last month we began a series of editorials hi-lighting products in each category that conform to the new way of doing things on the Mac, required by MultiFinder, AppleShare, new menu types, and the Macintosh II. We offered Write Now as an ideal word processor that meets our expectations for Mac software under the 'new rules of the game'. Except for the fact that Write Now blasts it's ruler on the desktop, instead of in a window, Write Now conforms very well to the MultiFinder friendly design rules that software should follow. Before we look at the database category of software, let's review why Write Now is a good example of the new way of doing things:

• **multiple windows work correctly**

Write Now can open multiple documents in re-sizeable and moveable windows, one of the chief requirements for MultiFinder friendliness. Windows are very important in the new way of doing things, because the user must be able to make the window small and shove it out of the way or to another screen while his attention is temporarily diverted to another application. Fixed size or immovable windows are a no-no!

• **Single-minded tool**

Write Now does one thing only, and does it well. It does word processing! It is simple, straight-forward in use, has no hidden feature functions that are not immediately obvious and does it's job very reliably. It has two nice features, a built-in spelling checker that is very convenient to invoke, and an easy to use leading feature which is particularly important for Pagemaker, which can also import it's files directly and respects the leading setting of Write Now paragraph by paragraph. There is no 'purpose clutter' from a program that can't make up it's mind whether it's a word processor or a layout program.

• **Small size and cost**

Since tools must now share the resources, it no longer is considered 'fair' to be a big system hog and take all the memory (and the money!) with large, high-priced software. Write Now is inexpensive (under $100), fast and small (80K versus 357K for MS Word), so it can co-exist with other tools without requiring 5 megs of memory.

• **Easy to Learn**

There is a growing trend toward tools that come with six inches of documentation and 300K help files! The Mac look and feel, if properly implemented, should result in an obvious and easy program to learn and use. The Write Now manual is reasonable in length and the program uncomplicated, but flexible and powerful *for the task it was intended.* This is good tool design, the Macintosh way.

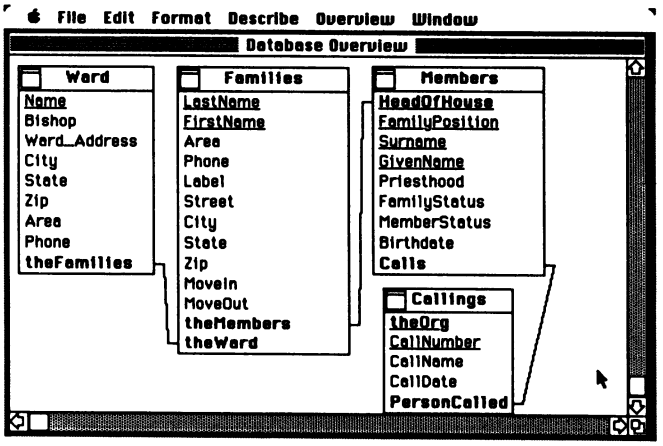


Fig. 1 The Reflex Plus Database window shows our Church Membership or Geneology database example

FileMaker Plus
The Best 'Flat' Database

In the application world of databases, Filemaker Plus has always been regarded as the best flat database product on the market for any computer. It too follows the new way of doing things. The very first version works on a Macintosh II; not many programs can make that claim. They have never had to revise the program to meet Apple's changing system! I don't know of any other significant Mac program that can say that. It is interesting that when Microsoft bought PowerPoint from Forethought, they didn't buy FileMaker Plus! Is it because FileMaker Plus is far superior to Microsoft File? FileMaker Plus is inexpensive, fast and small in size (260K), uses windows properly and does a single minded job very well and with great flexibility. Nearly anything can be imported or exported from the program. FileMaker Plus typifies the new way of doing things for a flat database product. What we have said for years is that the world needs a relational FileMaker. We think we have found it. Read on.

Reflex Plus
The Ideal Relational Database

Goldylocks sat on the Hypercard chair and said "This chair is too soft". Then she sat on the Fourth Dimension chair and said "This chair is too hard". Finally, she sat down on the Reflex Plus chair and said "This chair is just right", and she finished her database design and fell fast asleep.

Borland's new relational database, Reflex Plus, is the best

Macintosh database that conforms to the new way of doing things on the Mac. It is simple to use and operate, very flexible in design, low cost (only $275 compared to $500 for dBase Mac), and small in size (250K versus 730K for Fourth Dimension). It completely solves the generalized relational problem between arbitrary files in a flexible way that makes the generation of complex database applications a snap. We converted an Omni 3 database that took two weeks to design into Reflex Plus in two days, and that includes time to unpack and glance at the manual! On the third day, we had added more reports, fields, and files to the design that we never did get around to figuring how to implement in Omni 3. All the information in our Omni 3 database was easily exported and imported into our Reflex Plus design. An outline of the design for a Church membership system or a genealogy system, where families and members of families must be tracked, is shown in figure 1. This type of application is particularly good for checking out database products because it is so obviously relational.

Mac Friendly

One of the best features of Reflex Plus is the ability to use the Macintosh way of doing things to design the database file, record and field relationships. Opening a new database file, results in a graphic box in the overview window. Each entry in the box represents a field in each record. Relations between the fields of different records in different files are established by drawing a line from one field to another! What could be more Mac'ish or easier than this? The over-all visual effect of being able to always see the total database design is both stunning and effective. You immediately see new things to add or change in the design, so that very quickly you can create the most sophisticated relationships. Fourth Dimension also has this ability, but Reflex Plus is much easier to set up. In fact, Reflex Plus makes a great template designer for Fourth Dimension, but after quickly creating something in Reflex Plus, you may not want to bother porting the design to Fourth Dimension, unless you want to protect your design and sell it.

Export Function for Reports Needs Help

Reflex Plus has both an import and export facility, but the export function is slightly brain-damaged. It can't export a report to a text file arbitrarily. If the report is relational, that is it uses nested repeating collections, only one repeating collection at a time can be exported, in effect, flattening out the database. We think any database program worth it's salt should be able to take anything it can create on paper and send it to a text file so that reports can be merged into other corporate communications electronically, with page layout programs. This is one of the strong features of Omni 3 and we would like to see it added to Reflex Plus. Other than this small problem, Reflex Plus wins our award as the best Macintosh relational database, exemplifying the new way of doing things in a Multifinder universe. There doesn't seem to be any kind of database application we can't create with it in just a few hours. I can't even figure out how to get Fourth Dimension booted up in much less time than that!

Printing In Columns Supported

One tricky task we wanted to do was to print a report in columns. Loree Flescher of Borland showed us how to easily add a few fields to our report to compute the number of lines used by the nested repeating collections, and to divide the records into left and right side so that the report correctly prints records up and down in two columns. Figure 2 shows how this is done, by making a 'calculation report', and then a second, 'honest-to-gosh' report that copies the fields of the calculation report into double columns.

Make Any Database Print In Columns

A neat trick to make any database program print in columns is to modify the postscript code when the document prints. Press cmd-F just after clicking PRINT in the print dialog box and a postscript file image of your report will be created. Scan this file with a text editor and you should see two lines at every page break:

```
F T cp
%%Page: ? 2
```

By deleting these two lines on the even pages and inserting a translate command on the odd pages, after the cp command, your report will be automatically formatted into columns, when you dump the postscript to the LaserWriter:

Even pages: replace with

```
280 0 translate
```

Odd pages: use the following

```
F T cp
-280 0 translate
%%Page: ? 2
```

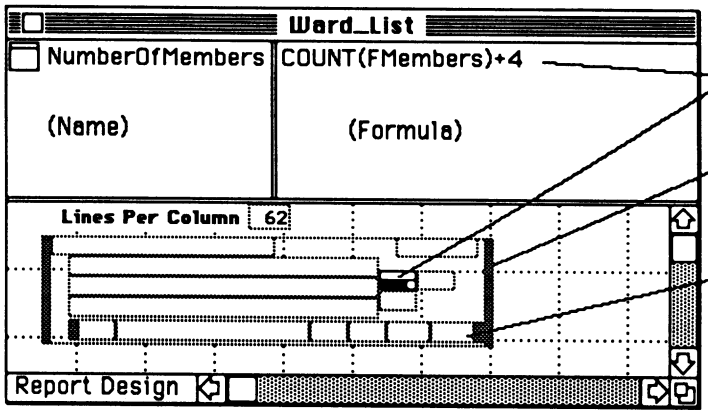
The cp command is 'LaserPrep' shorthand for a macro which is defined in the LaserPrep file. Within that macro is a show page function, which ejects the page. The translate command moves the origin to the middle of the page, so by removing the show page, the next page gets printed on the same page, right side. On the odd pages, you move the origin back, so the next page comes out on the left side. Simple! You might also want to adjust the %%Page command with the corrected count of pages as well as the last %%Page command in the file which has the total page count, now half what it was. It is amazing what a little postscript knowledge can do for you. (We would really like to see a definition article that explains most of the postscript macros contained in the LaserPrep file. This file can be dumped out by pressing cmd-K instead of cmd-F at print time. This would greatly help in decoding cmd-F postscript dumps.)

Get a Big Screen, or Multiple Screens!

One point of advice, and this probably applies to any database product, you should have a large monitor or multiple monitors if possible. Working with two monitors on a Mac II was a real joy in setting up Reflex Plus applications because there was room for the many windows created for the files, entry forms, and report forms, all of which are very flexibly implemented. In preparing this editorial, I moved the program to a Mac SE and the small screen felt very uncomfortable.

Things We Would Like To See

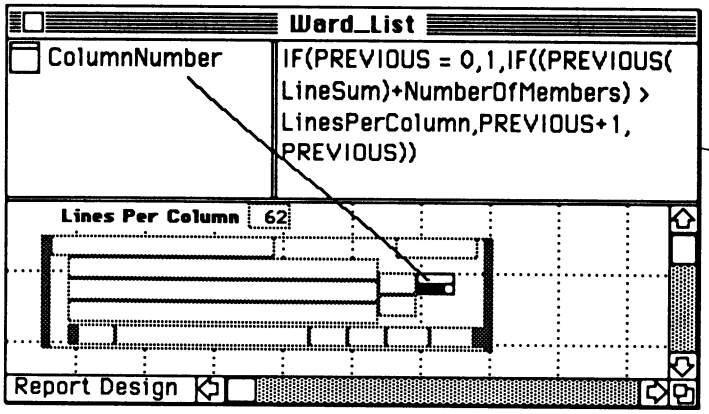
A few improvements we would like to see, besides the repair of the export function for reports already mentioned, is some



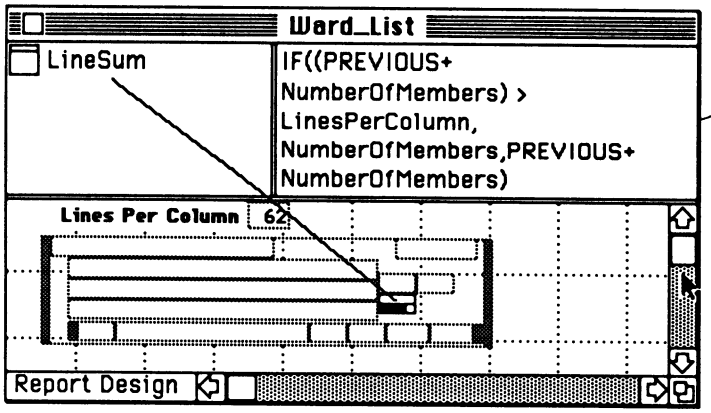
Step 1: Define a report to print out all the family records, with the family member records for that family nested within the family. This is a nested 'repeating collection' on the records of the family file and the linked records of the member file.

To get the report to print in columns, we define extra fields. The first is a constant showing how many lines per column we want, placed outside the family repeating collection.

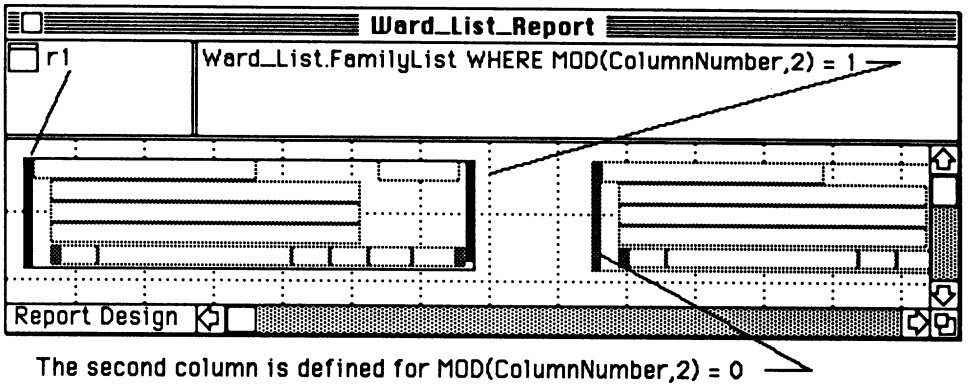
The second, shown selected here, is a count of the number of member records linked to this family. We add four to the numbe because our report has four printed lines per family for family information.



Step 2: The column number for the family record is computed modulo 2 from the total line count used so far in comparison to the lines per column.



Step 3: The total line count so far is computed by adding the lines of the current record (family + members) to the previous value and checking to see if we still fit on the page. If not, we keep the count the same, otherwise, we add these lines.



Step4: The 'real' report is made by duping the 'calculating report' and copying the fields on the left and right side of the page. Those families which have a 1 in their colmun number in the previous report will appear on the left; those families which have a 0 in their column number will appear on the right. The five 'calculating' fields on the dummy report are not printed.

Fig. 2 Making Reflex Plus Print Reports In Columns

additional built-in functions, particulary one to convert a string into a number. This seems oddly missing, since the converse function of making a number into a string is included. The other area of concern, and again this might apply to other products as well, is that the Apple date routines are used. As a result, dates are limited to something like 1904 to 2039. Well, my Grandmother is still living and she was born in 1899 so I can't enter her birthdate into a birthday file! Also, I plan to be around after 2039 so what happens then? A few more formatting options for dates would be nice also. We used a LOOKUP function to read the string label, 'JAN" for a month and use it in our reports, so it is possible, but it could also be built-in to the product as well.

Differing Philosophy

There are two approaches to building a general database program. You can make it programmable so developers can create commercial templates and sell them to vertical market applications. Or you can make it user friendly so the end user can constantly adjust, change or modify the design in a Mac-like friendly way. This eliminates the need for database programmers in many applications. Borland has taken the latter approach with Reflex Plus, while programs like Fourth Dimension and dBase Mac have taken the former approach. (Some might argue that you can do both in one program.) We think Borland is right on target with their approach. A well-designed database program should not have to be programmable to be flexible. There is however, a legitimate need for database designs that are programmable and unchangeable. For some applications, the ease with which you can change a Reflex Plus design may be a problem (a protect object menu item is provided, but no password protection). And it is not clear how suitable Reflex Plus would be for a multi-user application over AppleShare, where more than one Mac user must work on a common database at a time. It appears the product is best suited for the single user category. In any case, the program best exemplifies good use of the Macintosh user interface, and conforms to the expected Mac behavior and ease of use most of us love about the Mac. After using it, I am convinced I can create any database application in a matter of hours with this product. And with my busy life, the speed with which I can 'figure it out' and get the job done is very important. (Too bad MPW can't be like this product!)

Colorizer Restores CMD-Shift-3

Kenelm W. Philip
Fairbanks, AK

I think you owe an apology to Joel West (Palomar Software). In the November issue of MacTutor, you comment on page 12: "There is still no way to capture the screen image of a color display." On page 6 of the same issue is an ad for Colorizer, which mentions that this program will "Record full-color screen images to disk or printer". Apparently the left hand knoweth not what the right hand is doing...

As a satisfied user of Colorizer, I can vouch for the fact that the package contains an FKey which will dump a color screen to disk as a PICT file. To my mind, this FKey alone is worth the price of the package. *[We agree! -Ed]*

Finally, a tip that may be worth passing on to those Mac II owners who happen to have more than one monitor hooked up: If you hold down the Command key when you drag the grow box on a window, you can overcome the built-in limits on window size that many programs have (for example: Reflex+), but fails for a few (for example: MacWrite 4.6, whose window will snap back to the built-in limit as soon as you release the grow box). *[If Reflex Plus has any limitation on it's window size, we can't see it on an Apple color monitor! -Ed]*

68020 versus 80386

William Clodius
Los Alamos, NM

I have a few comments on the recent discussions that have appeared in your magazine on the relative performance of the 80386+80387 compared with 68020+68881. I believe your readers are incorrect in implying that a flawed design necessarily implies lower performance. The problems with the 80x86 design include the use of segmented memory, a smaller number of registers, and incompatible memory modes for different sizes of memory space (the 80286 protected mode problem). These problems complicate the design of programs, particularly of large ones, and make it impossible to write a program for earlier chips that take advantage of the larger memory space of subsequent chips. The need to implement several memory modes on the 80386 also increased the complexity of the design, and hence may be a contributing factor to Intel's well publicized difficulty in manufacturing the 80386.

However, while the above problems complicate a programmer's task and increase the cost of a working 80386+80387 system, they do not directly affect the performance of the system. For systems of comparable complexity, ie. two CISC designs, the performance should be roughly proportional to the MIPS rating. Because the 80386 includes the MMU on chip and uses a higher transistor density to implement faster algorithms for some instructions, it takes one to two less clock cycles than the MC68020 with MMU to perform an instruction so that it is effectively 15-33% faster than the MC68020 at the same clock speed. This increased performance is not unreasonable given that the 80386 appeared well over a year after the MC68020, and its designers could both study the MC68020 for ideas and take advantage of improvements in manufacturing technology. However, it is significantly less than that suggested by the recent notorious Byte articles.

The above disagrees with analyses by some of your readers that assume that the 80386+80387 differs in performance from the 80286+80287 only by a factor proportional to their clock speed. However, by the same reasoning a 16MHz MC68020 should be only twice as fast as an 8MHz MC68000, instead of about four times as fast. In such comparisons it should be noted that a 32-bit bus in and of itself will give an additional factor of two improvement for 32-bit operations over a 16-bit bus (ie. floating point operations), and can give additional performance increases for 8- and 16-bit operations by using the registers as caches and by sometimes performing operations in parallel. Further performance increases occur for newer designs because the increased transistor density allows the implementation of faster algorithms and special techniques, such as putting the MMU on chip, to reduce the number of clock cycles per instruc-

tion. Such techniques allow the MC68030 to be about 50% faster than the MC68020. The increased transistor density also allows the use of new instructions, such as the transcendentals implemented on both the 80387 and MC68881. As a result, your readers should not predict the performance of an 80386 system using an 80286 compiler, although some of them have attempted to do so.

Finally, a few assorted points. Neither the 80387 nor the MC68881 is the fastest available floating point unit. The Weitek processors, for example, are much faster than either chip at floating point operations. Although I believe the 80386+80387 is faster than the MC68020+MC68881, the performance difference between the two groups of chips is not significant compared with reasonable differences in compilers, operating systems, and bus characteristics. However, if Motorola's performance goals for the MC68030+68882 are achieved, then their performance should be much faster than the Intel chips for reasonable variations in other aspects of the system design, ie. a Mac "III" should decisively outperform any 80386+80387 system.

Does Font/DA Move 3.6 have any bugs?

Peter Trinder
Berkshire, England

I have a technical question to ask in respect of MultiFinder. I have received from Apple Developers group (the UK version of APDA), the UK localized version of MultiFinder and I have encountered some weird problems using the 3.6 Font/DA Mover. The main trouble seems to be that when I remove more than one Font, the Font/DA Mover fails with an error (-108) and the font file becomes damaged (or if I am removing from the system, the System is damaged). I only seem to get this problem using a MacPlus with more than 1Mb of memory, in my case it is a TurboMax. A friend has had this problem on a standard MacPlus with 4 Mb! I checked with Apple UK and they told me on 16th of November that the release of MultiFinder [in the UK?] had been delayed. Does anyone know if this is the reason? I called Steve Brecher about a week before I heard of the delay and he was puzzled but could not offer a solution.

[Multifinder is shipping here and we have not seen this problem with the Font/DA Mover. However, you cannot 'edit' your system file while running Multifinder. You must return to the normal Finder before you move fonts and DA's in and out of the system. The only problem we have verified is that which we reported last month: some fonts may need to have their flag word checked in their FOND resource to make sure it is $6000 to work correctly under Pagemaker 2.0a. We also noticed that Edit has problems with more than 24 fonts under the new system. -Ed]

MultiFinder Patch for PageMaker

He did however give me a patch for PageMaker 2.0 & 2.0a which will allow it to switch correctly in MultiFinder; the problem is that PM only recognizes 31 entries in the Apple Menu. Switching in MultiFinder looks for the entry in the Apple Menu; if it is more than 31, nothing happens. To fix this, search for 0102 011F and change it to 0102 01FF. This will allow PageMaker to honor 255 entries. Thanks Steve, you are great!!

Riccardo Ettore came to the MacUser show in London (10-12 Nov.) and gave me a disk with his new **Sound Mover** Package. This has a converter for taking SoundCap files and producing .snd resources for both HyperCard and the Mac II. He has also included a Sound Mover which is designed like the Font/DA Mover and this will install sound resources into the system. Of interest to MacPlus and SE owners is his IBeep2 CDEV which is placed in the system folder and appears in the Control Panel (System 4.1 or higher) and allows the Mac II beep sound facility on these machines. He has included an assortment of sounds, I have the Beatles HELP! Makes a nice change. It should be on CompuServe by now. *[Note to authors: We would dearly like to publish some type of sound mover CDEV that explains how to add sounds to the Mac II 'library' of exception beeps. -Ed]*

Draw over the Menu Bar in Basic

Anthony J. Oresteen
Batavia, IL

One of my biggest frustrations with ZBasic for the Mac was I couldn't control the ENTIRE screen. I wanted to draw over the menubar but just couldn't. Well, thanks to a hint from Ann Arbor SoftWorks (FullPaint) I figured it out.

The solution is simple (as most are once you know them!). You can draw visable windows only on the defined region of the desktop. When the Window Manager is initialized, it draws the desktop and empty menu bar. It stores the desktop as a region and the global variable GrayRgn (&9EE) holds a pointer to it. By expanding the desktop region to the full screen size, you can draw windows over the full screen.

Below is a program that draws a full screen, inverts it to BLACK, and then restores the desktop. NOTE that it will work with any size screen as it checks the global screen.bits for the screen size.

Two bugs remain. First, the lower corners of the desktop do not return to "round rect" types. They are square but should be rounded. Secondly, when scrolling text in the black window, you get funny white bars. Any help on solving these minor bugs would be appreciated. Thanks.

```
REM THIS DRAWS WINDOW OVER MENU BAR
REM A. J. ORESTEEN AUG 1987
REM PUBLIC DOMAIN USE FREELY
REM THANKS TO ANN ARBOR SOFTWORKS ("FULLPAINT") FOR HINT!!
REM USE ZBASIC 4.0 [or later? -Ed] !!!
:
BREAK ON  :  CLS  :  COORDINATE WINDOW  :  WINDOW OFF
:
VSIZE%= PEEK WORD(PEEK LONG(&904)-116): REM VERT SIZE OF
SCREEN
HSIZE%= PEEK WORD(PEEK LONG (&904)-114): REM HOR SIZE OF
SCREEN
GRAY&=PEEK LONG (&9EE): REM GETS GRAY REGION OF DESKTOP
CALL OPENRGN : REM SAVES REGION DATA
OLDDESKTOP&=FN NEWRGN : REM GETS NEW HANDLE FOR REGION
CALL COPYRGN (GRAY&,OLDDESKTOP&)    : REM GETS A COPY OF
STANDARD
DESKTOP
CALL SETRECTRGN (GRAY&,0,0,HSIZE%,VSIZE%): REM SETS DESKTOP TO
                                          FULL SCREEN
WINDOW #1,,(0,0)-(HSIZE%,VSIZE%),3  : REM DRAWS MAX SIZE
WINDOW
PRINT "BIG WINDOW!"
:
DELAY 1000
:
WPTR&=WINDOW (14) : REM HANDLE TO WINDOW
VISRGN&=PEEK LONG (WPTR&+24)    : REM VIS RGN OF WINDOW
```

```
CALL INVERTRGN(VISRGN&)    : REM INVERTS SCREEN TO BLACK
CALL BACKCOLOR (33)    : REM BLACK BACKGROUND COLOR
CALL FORECOLOR (30)    : REM WHITE FOREGROUND COLOR
CLS: BEEP : TEXT 0,18,0,0
PRINT : PRINT "BIG BLACK WINDOW!"
:
DELAY 1000
:
WINDOW CLOSE #1    : REM CLOSE FULL SCREEN WINDOW
CALL BACKCOLOR (30)    : REM WHITE BACKGROUND COLOR
CALL FORECOLOR (33)    : REM BLACK FOREGROUND COLOR
CALL COPYRGN(OLDDESKTOP&,GRAY&): REM RESTORES NORMAL DESKTOP
                                 REGION
CALL PAINTRGN (GRAY&) : REMDRAWS REGION
CALL DISPOSERGN (OLDDESKTOP&)  : REM DON'T LEAVE A MESS IN
                                 MEMORY
CALL DRAWMENUBAR  : REM RESTORES MENU BAR
:
WINDOW #2,"FULL SCREEN",(20,40)-(HSIZE%-20,VSIZE%-20),257
TEXT 4,12,0,0
PRINT:PRINT "MENUBAR IS NOW SHOWN AGAIN!!"
BEEP
PLOT 40,80, TO 200,200 : CIRCLE 300,200,40
:
"PAUSE"
GOTO "PAUSE"
```

Hypercard Questions
Scott Vore
Indianapolis, IN

A few questions please:

1) In Hypercard is there any way to call the SF GetFile dialog when importing text or do I have to specify the Filename each time? (I guess I could use CN XCMD)
2) In Hypercard is there any way to use application defined events like event #13-15 in the Event Manager?
3) The October issue of MacTutor (pg. 7) spoke of a button to import text in the button ideas stack. I sure can't find it. Any ideas?
4) Any idea why a company that calls itself (or product) "Lightspeed" won't whip an update for Lightspeed Pascal for the Mac II without a check (not plastic) and then takes over two weeks to do it? Beats me.

[The latest LS Pascal update, 1.11, is available on AppleLink (I think), Compu-Serve, and direct from MacTutor at no charge to our customers. A new version should be announced at the Expo this month. More next time, if they release it. We have had a number of people contact us about writing regular Hypercard articles, but nothing yet. We would like to have an on-going hypercard column, if someone has the 'stick-to-it' fortitude of a Dave Kelly or Jörg Langowski to pull it off. -Ed]

Volume 4 Number 2

Making WriteNow MultiFinder Friendly
Mr. Sleazy Hacker
Boulder, Colorado

Here's a very sleazy patch to get around WriteNow's habit of blasting a ruler across the entire screen (assuming a Mac II sized screen of width 640 pixels). The patch was applied on a Mac II with B/W monitor, and version 1.0 of WriteNow.

Using Fedit or equivalent, search for:

```
A91B 2F0C 2055 3F28 FF92 5557 3F3C 002C
```

And replace with:

```
A91B 2F0C 3F3C 0258 4E71 4E71 3F3C 002C
```

There's one occurrence of the pattern. What we are doing here is replacing:

```
movea.l (a5),a0
move.w  ff92(a5),-(a7)
subq.w #2,(a7)
...
_SizeWindow
```

with:

```
move.w #$250,-(a7)
nop
nop
...
_SizeWindow
```

In other words, the code fragment that calculates the width of the ruler as (**screenBits.bounds.right -2**) is replaced with one that supplies a constant value of $258 (decimal 600). *[Note to developers: they never should have used screenBits.bounds for this calcuation. Since their ruler is in it's own window, as David Dunham points out below, it should have been defined as a fixed size window or better still, growable. -Ed]* The largest argument that **subq** will take in this form is 8, hence the substitution with a constant. Of course, once you've done this screen-size independence flies out the window, so don't do it to an original...(You can find this peice of code with TMON by putting a trap intercept on GetNewDialog, launching WriteNow and waiting for the first intercept. GetNewDialog is immediatly followed by a MoveWindow and the SizeWindow shown above.)

When the ruler is shortened by 40 Pixels in this fashion, there's just enough room for part of the top Finder icon to peek through.

Since this trick is so sleazy, I think I'll request anonymity of you decide to publish it! *[Granted! -Ed]*

By the way, there's one MacWrite feature that I especially miss under MultiFinder when using the newer word processors - its ability to put formatted text on the clipbord (the 'MWRT' scrap format). Microsoft Word, WriteNow (at least the version I tested) don't do this, so you're cutting and pasting plain text. Since PageMaker seems to understand this scrap format, that's a pity. The 'MWRT' format isn't perfect — you can't pass leading values, specify baseline offsets or odd-ball styles like strike-through, but it's better than nothing. I'm hoping that there will be some improvement in this area in 1988. *[We agree! The whole point of developing a new Text Edit was to support a new formatted text type for the clipboard, but no one is using it yet! We encourage developers to standardize on support for the new formatted text type in cutting and pasting across applications. How about an article to encourage this someone? -Ed]*

Reflex Plus Comments
Kenelm W. Philip
Fairbanks, AK

It was with great interest that I read your editorial on Reflex Plus in the January '88 issue of MacTutor. After reading it, however, I wrote the enclosed letter in the attempt to urge Borland to fix two defects (one annoying, one serious) in the existing versions of reflex Plus.

© The Definitive MacTutor, Vol. 4

P.S. If you will reread my previos letter (which you printed in the January issue), you will see that I was talking only about Mac II systems with two monitors when I mentioned the "COMMAND-Grow_Box" method of expanding windows. Both Reflex Plus and MS Word, for example, will not allow you to expand a single window to cover two mmonitors without using this handy trick.

I am glad to see your editorial on Reflex Plus in the January issue of MacTutor. I have been using Reflex intensively since it first came out from Singular Software as Interlace in 1085, and I Have found the program to be almost ideal for my purposes (which involve maintaining data on a research collection of Alaskan insects). However, I think your readers should be aware of a couple of disturbing problems that have emerged, one involving the use of Reflex with Apple System versions 4.1 and later, and the other involving Reflex Plus.

Date Formatting Limited

Problem # 1 has to do with the formatting of fields declared to be of type 'date'. According to the manual (for both Reflex and Reflex Plus) there are three date formats:

- short: 12/31/84;
- long:December 31, 1984;
- abreviated: Dec 31, 1984.

Unfortunatly, this is not the case with System 4.1 or 4.2, where both the Long and Abbreviated formats add the *weekday name*, becoming:

- Monday, December 31, 1984
- Monday, Dec 31, 1984

I have verified this problem for Reflex 1.00 and 1.01, and Reflex Plus 1.00 and 1.01, with Systems 4.1 and 4.2.

For my work on insects, dates are very important but weekdays are not. It was disconcerting to find that all my carfully formatted reports were fouled up by the addition of an extraneous and unwanted weekday name in date fields as soon as I upgraded to System 4.1. It occurred to me that the problem might very well lie in the International Resources in System 4.1, and that it might be possible to hide the weekday by activating the 'supress dayname' byte in those resourses. I ran into some problems trying to do this, so I sent a request on AppleLink for help with this operation. To my vast astonishment (remember that Apple and Borland are currently engaged in a joint promotion campain for Reflex Plus!) the reply was, in essence: "We do not recomend using ResEdit on applications. We suggest you try another database." I had not even suggested using ResEdit on Reflex Plus.

I then wrote Borland. They replied that the problem lay entirely with Apple's System software. Somehow, I was expecting them to say that— But the problem remained. It also looked as if the technical support people at both companies were totally unaware of the Apple/Borland joint promotion campaign.

I was able to elicit a workaround for the problem. If you declare the field to be of type 'text' in your report, you can constuct a formula which will put the date into any form you like (without weekday names)— and Borland was kind enough to supply me with such a formula. I reproduce it below for the benefit of other people who have run into this other wise intractable problem. Assume that your date field is named 'WHEN', and you would like dates in the format: 31 Dec, 1984. Use the following formula in the report, where the new date field is a text field:

```
DAY(WHEN)" "&IF(MONTH(WHEN)=1,"Jan"IF(MONTH(WHEN)=2,"Feb",
IF(MONTH(WHEN)=3,"Mar"IF(MONTH(WHEN)=4,"Apr",
IF(MONTH(WHEN)=5,"May",IF(MONTH(WHEN)=6,"Jun",
IF(MONTH(WHEN)=7,"Jul",IF(MONTH(WHEN)=8,"Aug"
IF(MONTH(WHEN)=9,"Sep",IF(MONTH(WHEN)=10,"Oct",
IF(MONTH(WHEN)=11,"Nov","Dec")))))))))))&", "&YEAR(WHEN)
```

A slightly different version of this formula is available from Borland as Tech Info note #370, "Truncating the day of the week", date 28 september 1987. *[We used a better refinement of this method by setting up a look-up table file to associate the month name with the month integer. This is a more elegant approach than the massive If...Then..Else statement. -Ed]*

We are not out of the woods yet. A date field in a report so formatted will not sort by date. If you need to sort your data in chronological order, and you have Reflex(rather than Reflex Plus) you are out of luck. In Reflex Plus you can do the following: create one report in which the date field is of type 'date', and sort that report on date. Create another report in which the date field is declared to be of type 'Text', using the formula above—and then call the repeat rectangle from the first report in the second report (see page 184 in the Reflex Plus manual: "accessing Data in other reports"). This cumbersome workaround will take care of the problem—but it's a kludge! Apple and Borland should get together and solve this problem. *[Borland is right in saying that Apple modified the International date routines, which many companies use as is to avoid having to duplicate functions already done by Apple. But Borland should add to their built-in function list to include more date formatting functions. However, this is minor compared to the limitations of the exportation of reports I reported last month! My brain-damaged report exportation fix should be at the top of their 'things to do' list! -Ed]*

Header and Footer Problems

Problem #2 is one that Borland introduced with Reflex Plus (both version 1.00 and 1.01). Something went wrong with the formatting of headers and footnotes in reports, and they now take up the entire space within the header and footer boxes even if they are only one line in length. that in : a one line header will be separated from the body of the text by about 1 3/4" of blank space; and a one line footer will have about 1 3/4" of blank space between itself and the end of the page, leaving about 6" of space for the text of the report. I have not yet verified this problem with all concievable configurations of hardware, but it has turned up with the following: Macintosh+, System 4.1, 15" ImageWriter; Macintosh II (2 MB RAM), System 4.2, 15" Image-Writer; Macintosh II (5 MB RAM), System 4.2, LaserWriter-from which I conclude that this is almost certainly a universal problem with Reflex Plus.

I called Borland about this situation, since I found Reflex Plus essentially unusable when it came to printing my data in properly formatted reports. The first response I received from Tech Support was that I had run into a problem that sometimes

occurs when you take a Reflex database into Reflex Plus, and that I should throw away the existing headers and footers and re-install them from within Reflex Plus. That accomplished nothing-so I created a new test database file from within Reflex Plus, and, as I suspected, the exact same problem turned up. I called Borland again, and Tech Support now (reluctantly) admitted that they had received a number of angry calls about this problem. Nothing was said about when or if a fix would be made available.

I find it impossible to believe that beta testing did not expose this problem (unless Borland does not do beta testing). I am thus forced to the unpalatable conclusion that Borland released Reflex Plus unfinished, knowing it was not in fact yet suitable for the business environment for which it had been targetted. I had written Borland earlier to inquire if the major price increase in Reflex Plus would be accompanied by a proportional improvement in their user support, since their previous policy had been not to notify registered owners of upgrades. They assured me that I would be seeing a totally new level of support for Reflex Plus. It seems to me that one of the first steps in user support might be to provide customers with a business database that could print proper headers and footers...

I totally agree with the main thesis of your editorial-Reflex Plus is potentially a fantastic program, and would be my preferred database environment if it could only print decent reports. For the time being, however, I have archived Reflex Plus and returned to Reflex, which can handle headers and footers. *[If my export problem were fixed, then you could export the report to Pagemaker and format it anyway you like! -Ed]*

Color Scanning
Douglas K. Beck
Los Altos, Ca.

As to things that seem to work on the Mac II: The PCPC IHD 144 internal disk installed the first time with just a screwdriver, came up with no trouble and has performed with speed and efficiency, 10 hours a day and more, for a couple of months now.

The RasterOps (Cupertino) 24 bit RGB controller is also performing up to expectations. I have written some display programs for it and it works up to the book with few misunderstandingsl I scanned some 1024 x 768 pictures from some 8x10 color positives that look fine. One picture of the Stanford Chapel, full of textures and fine detail, a bouquet of flowers and some color test patches, including the Macbeth color test pattern. I am driving a Mitsubishi HG6905BK 20 inch color display with the RO board. I am less enchanted with the monitor. All common adjustments require dismantling the thing since they are buried on the main pc board. A long insulated screwdriver is required, and you must fish down past the second anode terminal to get to them. Sigh.

I have received the National Instrument DMA controller, but have not tested it. Looks interesting, and uses an Intel chip to do the transfer thing.

Still to be received are the National Semi 16 Megabyte board and the Optronics 400 meg WORM unit. I will report how they work if desirable.

One thing that is causing pain right now, is having to convert from Megamax C to Lightspeed C to gain access to the color stuff on the II. Megamax is not going to support their Mac product any longer, and have orphaned the product just as it is coming of age on a machine that has the speed and multitasking capability to make it viable. Lightspeed is a nightmare. The inconsistent handling of strings is pitiful. With Megamax, all strings were C strings and the compiler took care of all the conversion work. Lightspeed lets you guess, Pascal or C this time? Bah. Can't someone lean on Megamax and get them back in the game?

Spleen vented. Flame off. *[A better approach would be to lean on Think to enhance their very popular C compiler to make string handling easier. -Ed]*

I agree on WriteNow. I recommend it without reservation for all but the heaviest of duty tasks.

Incidentally, making mailing labels with HyperCard has been a pain. The ImageWriter is just not up to the task. However, if you get some Avery 5331 liquid toner copier labels (for use with the Cano Engine) the LaserWriter does a very creditable job. Looks neat on the Christmas Cards. *[I use a very solid NEC Spinwriter for mailing labels for MacTutor's 13,000 cirrculation. Apple simply does not make a business printer that can do labels reliably in the volume we need. The Spinwriter, however, is a truck. It can run all night and never jam. But it was a pain to get it working with the Mac! -Ed]*

Speaking of Christmas, a Merry one to you and the very best for the New Year.

Volume 4 Number 3

Spreadsheet / Word Features
Steve Millina

In the January issue 1988 letter column, Cary Maraish asked how to do spreadsheet-type windows fairly easily . I just did a one-page spreadsheet, and I found that the list manager does a great job. Set up a rectangle for the cells, and feed LInit the number of rows and columns, but FALSE for both scroll bars. Since this was a true spreadsheet, actual text entry was through a text edit box at the top of the page, with the text being transferred to the list with LSetCell only when the user indicated entry was complete. For convenience, I used a dialog window so that the text edit stuff was taken care of directly. You do need to bear in mind that the default text list definition proc justifies left , so the programmer has to insert the appropriate number of spaces in front of the numbers to make them line up.

If you really want entry to be made at the selection, you might try a Dialog window with a grid of edit text boxes. (I tried something similar with a grid of stat text boxes and it worked fine).

Since I'm already writing, I want to take issue with your evaluation of WriteNow. I tried desperately to like WriteNow for many of the reasons you mention, but in the end I went back to Word 3. The most important determinant was the fact that setting line spacing was a slow and painful process, because I use lots of single spaced paragraphs (for examples and quotations) in the midst of double spaced text and the switch took forever. So does setting the ruler.

I even tried macros. At the time I didn't have Quickkeys but

I did have Touch-n-Go, which is usually great. I tried to make it move the indent cursor. Fat chance! WriteNow doesn't look at the mousedown and mouseup events once it knows what's going on and the indent moves sort of randomly. I guess it reads the mouse button directly but that is not the kind of programming you should commend

Word, on the other hand, may be mammoth, and may have any amount of code wasted on oddities (such as outlining that only an IBM PC could love) but a lot of the add-on touches are incredibly useful. Styles lead the list. Also high is the command-= calculator that calculates whatever is selected and makes the answer directly available for pasting. I know you can do the same thing with some calculator DAs but the difference in speed and convenience is simply amazing. In a lot of respects, Microsoft did what good programmers are supposed to do, it found out how its targeted users work.

Big Arrays for C

Jack McCrae
Albuquerque, NM

The enclosed program would probably be appropriate for the inclusion in the letters section of your most excellent magazine. At least I think it's kind of neat (my program that is). And if it wouldn't be too much trouble would you please send me a copy of of your author's kit.

Two limitations I have run into while programming in the C language are the 32k byte segmentation limit, and C's inability to pass variable sized multi-dimensional arrays to subroutines. I have found a simple solution to both of these problems. Arrays constructed according to the source code which follows act like 'normally' declared arrays; no special functions are needed to assign into, or read from these arrays.

The following example program illustrates a three dimensional case. It can be easily modified for two dimensions, or extended for four or more. The cost of this method is a small amount of extra space which is required for one or more arrays of pointers, and an extra initialization step to create these pointers. As an added bonus, rows of these arrays may be rearranged by simply rearranging the pointers to them, rather than all of their elements themselves.

To use this method to create an array that takes up more than 32k bytes of memory, delete references to the variable 'arrayspace'and replace with the line

```
'intermediate[i][j]=arrayspace[i][j;'
    with
'intermedate[i][j]=malloc(sizeof(double)*kdemension);'
```

where kdemension is the number of elements in the outermost dimension.

This swaps the 32k total limit for an array for a 32k limit on each dimension.

The following compiled with either Megamax C, v 2.01b. or Lightspeed C v 2.13.

```
#include<stdio.h>

extern long tickcount();
long t;

    usearray(array)
    double**array[];
    {
      printf("array[0][0][0]=%f/n",array[0][0][0]);
      printf("array[1][2][3]=%f/n",array[1][2][3]);
      printf("array[2][3][4]=%f/n",array[2][3][4]);
     printf("array[9][9][9]=%f/n",array[9][9][9]);
    /*note that the above arrays appear to be accesed normally.
Had a pointer to a 'normal' array (i.e. arrayspace[]) been passed
instead of **array[] it would have been necessary to use a
construction like arrayspace[i*100=j*10=k] to access array
elements.

    */
    }
    /*this routine sets up 'array' to index elements of
'arrayspce' by double indirection */
    initarray(arrayspace,intermediate,array)
    double arrayspace[10][10][10];
    double *intermediate[10][10];
    double **array[10];
    {
      int i,j,k;
      for(i=0;j<10;j++){
        array[i]=intermediate[i];
         for(j=0;j<10;j++){
           intermediate[i][j]=arrayspace[i][j];
           for(k=0;k<10;k++)
             arrayspace[i][j][k]=100*i+10*j+k;
      }
     }
    }
    main(){
        int i,j,k;
        double arrayspace[10][10][10];
        double *intermediate[10][10];
        double **array[10];
      initarray(arrayspace,intermediate, array);
    asearray(array);
    while(!button());
    }
```

Higher Printing Authority

John Bartleson
Spokane, WA

First let me thank you for producing such a useful journal. I read it cover-to-cover every month and have found many useful items in its pages. MacTutor fills a unique place in the evolution of our favorite computer.

I am thinking of writing an article for MacTutor, but I need your advice on whether you would be interested in the topic. Here it is:

As I began to program the Mac, I had to tackle the problem of writing printing code. I eventually learned how to do it, but I think that the effort necessary to "do it right" was too high. There are several reasons for this. First, like much of the Macintosh system software, the Printing Manager is a relatively low-level interface to a process much different from that used on previous computers. Second, the Printing Manager as documented in Inside Macintosh is vague on many topics and needs to be rewritten. Finally, there is very little non-Apple documentation dealing with how to use the Printing Manager. The only documentation I've found is in Volume II of Dan Weston's The Complete Book of Assembly Language Programming. This has a good chapter on printing but it leaves out some of the fine points necessary for commercial - grade programming, such as error recovery.

Programming for Macintosh is more complex than is readily apparent from *Inside Macintosh*. You have to take care of draft copy looping, max pages in spool file looping, print-idle procedures, print traps vs. direct calls, etc. On top of this, you have to provide robust error checking that can handle both user cancellation, which may occur at any time, and Printing Manager errors.

I found that I was wrestling with these problems every time I wrote Printing code for a program. I wanted a higher - level interface to the Printing Manager, one that would be easier to use than the existing interface, but one that wouldn't limit me in getting any desired image on the page. This led led me to write the Generic Printing Manager, GenPrint.

GenPrint is a single subroutine that communicates with your calling program through a parameter block named GenPrBlk. GenPrBlk contains several flags and handles, and in addition it contains pointers to several co-routines. Co-routines are procedures within your calling program that do your specific print processing, GenPrint calls these procedures at appropriate times during printing. There are four co-routines, but the typical program will use only two. For example, you must provide a PrintPage co-routine to image the pages, but a JobDialog co-routine is needed only if you have a custom PrJobDialog you want to execute. Here's an example of typical use in pascal:

```
PROCEDURE MyPrint  (Demo of GenPrint call);
VAR ParamBlk:GenPrBlk;
(other variables owned by MyPrint would go here these can be
accessed within the co-routines.
PROCEDURE  GenPrint (Blk: GenPrBlk) ; EXTERNAL;
PROCEDURE DoInitCopy;
(Initialize pointers etc.  to print a copy)
END; (DoInitCopy)
PROCEDURE DoPrintPage;
 (   Draw a page in the printing GrafPort  )
END; (DoPrintPage)
BEGIN  (MyPrint)
(  Set the co-routine addresses in ParamBlk ).
ParamBlk.JobDialog :=0;
ParamBlk.SpSpace :=0;
ParamBlk.InitCopy := @DoInitCopy;
ParamBlk.PrintPage := @DoPrintPage;
        |
GenPrint  (ParamBlk) ;
        |
END; (MyPrint)
```

GenPrint is written in MPW Assembler, which is necessary so that it can set up the caller's A6 stack frame to allow the co-routines to access their enclosing procedure's variables. This means it can be called only from assembler or from languages (like MPW Pascal and C) that allow external assembler subroutines.

GenPrint greatly reduces the amount of coding necessary to do Macintosh printing. As an example, A commercial - class subroutine to print any Text Edit record, with support for fixed-space tabs, can be done in about 40 pascal statements.

While this technique may not be as exciting as an article on a new 256K ROM feature, I suspect a lot of people who haven't figured printing out would find it useful. And nothing on this topic has appeared before in MacTutor.

[Sounds great. Looking forward to seeing this. -Ed]

Dialog Boxes

David Dunham
Goleta, CA

I think Kirk Austin (Vol. 3, No. 12, p.73) has the right idea; "about" dialogs should be unobtrusive. Unfortunately, his routine prevents multitasking, with or without *juggler*(er,*MultiFinder*). It would be better to call SystemTask, to keep DA's multitasking, and GetNextEvent to keep background applications running. (Apple says that Systemtask is redundant under the Multifinder environment, but it's easier just to call it than to check your environment.) I also think a key should be able to cancel the "about."

```
while (true) {
        SystemTask();      /* keep clock ticking  */
        if (GetNextEvent(mDownMask | keyDownMask, &event))
            break;
}
```

If you prefer that the click which cancels the "about" be passed on to the appropriate program (remember the Mac has always multitasked —the user could be clicking in a desk accessory window to activate it), you can use something like the following:

```
while(!EventAvail(keyDownMask | auto =KeyMask  |  mDownMask,
&Event))
   SystemTask();
   /*Remove mouseDown if in our window */
   if (event. what ==mouseDown){
      GlobalToLocal(&event.where);
      if (PtInRect(event.where,&window->portRect)){
         GetNextEvent (mDownMask,&event);  /*swallow event*/
      }
   }
```

This code comes from my desk accessories, which have "transparent" title windows. I call them transparent because you can just start typing or mousing without having to actively cancel the window— events pass right through.

Volume 4 Number 4

But Can It Do Columns?

For several years, I have been interested in a relational DataBase application that keeps track of families and family members. I started out on an Apple II, using various DataBase products, whose names now escape me, to keep track of Church membership lists. Such an application is relational since you must track both the family information, and the member information for each member that is linked to that family. Since the Mac came out, I have tried various Mac products and have learned a lot about the problem of real world processing on the Mac.

One thing I have discovered is that nearly every Macintosh DataBase is incapable of formatting a report of families and family members in the same double column format we use for MacTutor, with a hairline box around the page and a line down the middle. The problem is the products cannot determine how

© The Definitive MacTutor, Vol. 4

many members are in each family and how many families will fit in the column before moving to the top of the next column. So I offer this DataBase challenge to any Macintosh DataBase product to see if they can create such a report. The winning product gets a **free ad** in MacTutor and a glowing editorial review. Here are my results so far.

I first tried **Helix**. However, I quickly became buried in icons and gave up after a few days. Then I tried **MacLion**. But they went out of business before I even got started. Next came **FileMaker**. Although this is a flat DataBase, I was able to accomplish the goal by setting up the file with both the family and members of family combined into a single record for each family. When I needed reports dealing only with members, I used a program written in Basic to "flatten" the DataBase and extract the members without the family information. The very good export and import functions of Filemaker make nearly anything possible. By placing the family report in Pagemaker, I was able to format it in columns. However, all this "pre-processing" got tiring after a few months, so I moved on.

Next came **Omni 3 Plus**. However, it soon became apparent that the relational aspect of Omni 3 had been tacked on to what was once a flat DataBase product, so that working with multiple relations between files was very difficult and not at all obvious. Although I was told there is a technical note for programming Omni 3 to format in columns from a relational DataBase, the complexity of the whole thing discouraged me and I gave up on it.

My most recent attempt has been with Borland's **Reflex Plus**. As I reported in a recent issue of MacTutor, Reflex Plus can solve the formatted double column report requirement, at least so I thought! Reflex Plus is particularly easy and nice when it comes to setting up multiple related files and anything can be quickly changed at any time. I love it! Which is why I am so frustrated that it can't format a page in double columns correctly. Oh you can set up the necessary calculations to group the families and members into a double column listing, but the page break option is brain damaged and can't figure out when to move to the top of the next page when using two independent repeating collections. As a result of this bug, you can't keep the family members grouped together without a break at the bottom of each page. And because the export function is also brain damaged, you can't export a report with nested repeating collections into Pagemaker and format the double columns there! So you are just stuck! So close, yet so far! (It also lacks the ability to make the box around the page and line down the middle and have them repeat on each page!)

That leaves **dBase Mac** and **4th Dimension** as our next candidates. Stay tuned!

Mouse-up Problems
Delmer A. Johnson
Castle Rock, CO

A few weeks ago I encountered a problem similar to the one described in Mike Steiner's Mousehole Report posting. I couldn't get icons in the finder to open when I double clicked. I could select them with a single or double-click and open them using the File menu or a command key. I discovered that the problem stemmed from another program which masked out mouse-up events. Evidently this is what the finder looks for when the user double-clicks on an icon.

Inside Macintosh II-70 warns against using SetEventMask for any purpose other than enabling key-up events. Unfortunately this advice is not heeded in the original edition of Macintosh Revealed vol. II, where mouse-up events are disabled (p. 26). The next time this problem arises, try writing a short program calling SetEventMask with EveryEvent. *[I've also seen this problem. Thanks for finding out what causes it. -Ed]*

Changing the subject, I would appreciate an article explaining how to display a text file longer than 32k using Textedit. Can a person connect several Text Edit records together?

Cricket Graph Data From Basic
Steven Leach
Santa Clara, CA

I greatly enjoy MacTutor, and have written for your magazine several times, but I must take exception to the article "The Workstation Mac" by Paul Zarchan in the February 1988 issue.

Besides needing a spelling checker, he must not be very familiar with MS BASIC on the Macintosh™, since I have been writing Cricket Graph™ compatible files for as long as the product has been commercially available. They specifically included a section of pascal code to write the tab delineated files, which was extremely simple to write in BASIC as well. When the 1.1 version of Cricket Graph became available, I did find that my BASIC program had to be modified because of the tendency of the new version to treat the leading space in the BASIC file string for the negative sign as an alpha character rather than a numeric character. This problem was easily fixed using the print using statement in BASIC.

When my data files became too large to load quickly using the tab delineated format, I simply called up Cricket and they kindly sent me a single page description of the fast file format. This format I have incorporated into all my BASIC diagrams, some of which have been available commercially and as shareware for some time.

I have included honest-to-gosh subroutines which can be used for exactly these purposes. So you can see that there is no problem in writing Cricket Graph compatible files. It is this type of misunderstanding and lack of curiosity that has relegated BASIC and FORTRAN to the "engineering" community, or to the "toy" factory. I personally program in Pascal, C, FORTRAN or BASIC which ever will do the job more simply, quickly and efficiently, although with the interpreter, compiler combination it is becoming tougher to justify not using BASIC when I don't need special data structures.

```
LIBRARY "MS Tools"
'This is just a sample routine that fills an array
'Saves it to Disk as a Cricket Graph ™ file
' and then retrieves data and displays it on the screen
N.col% = 5
DIM test.data(N.col%,10),test.title$(N.col%)
RESTORE
FOR indx% = 0 TO N.col%
    READ test.title$(indx%)
```

```
NEXT
DATA "Sample Ø","Sample 1","Sample 2","Sample 3","Sample
4","SAmple 5"
FOR cindx% = Ø TO N.col%
    FOR rindx% = Ø TO UBOUND(test.data,2)
        test.data(cindx%,rindx%) = rindx%+100*cindx%
    NEXT
NEXT
file.name$ = FILES$(Ø,"Name for File ?")
IF LEN(file.name$) < 5 THEN END

CALL save.Cricket.tab(test.data(),test.title$(),N.col%,
file.name$)
CLS
ERASE test.data,test.title$
DIM test.data(N.col%,10),test.title$(N.col%)

CALL
open.cricket.tab(test.data(),test.title$(),N.col%,file.name$)
FOR cindx% = Ø TO N.col%
PRINT  test.title$(cindx%)
    FOR rindx% = Ø TO UBOUND(test.data,2)
    PRINT  test.data(cindx%,rindx%),cindx%,rindx%
    NEXT
NEXT
END

'  This subroutine reads in the Data ffrom a Cricket Graph™
Fast Format file
SUB Open.Cricket(array(2),array.title$(1),N.col%,file.name$)
STATIC
    watch% = 4
    changecursor watch%
        OPEN file.name$ FOR INPUT AS #1

        tmp% = ASC(INPUT$(1,1))
        tmp% = ASC(INPUT$(1,1))
    IF tmp% <> N.col%+1 THEN
        INITCURSOR
        msg$ = "This file is not compatible with this Data
Array"
        CALL errormsg(msg$)
        EXIT SUB
    END IF
    FOR indx% = Ø TO N.col%
        tmp% = ASC(INPUT$(1,1))
        array.title$(indx%) = INPUT$(tmp%,1)
        IF tmp% < 15 THEN tmp$ = INPUT$(15-tmp%,1)
        tmp% = 256*ASC(INPUT$(1,1))
        t.col% = tmp%+ASC(INPUT$(1,1))
        count% = Ø
        indx2% = Ø
        WHILE count% < t.col%
            LINE INPUT #1,tmp$
            count% = count% + LEN(tmp$)+1
            array(indx%,indx2%) = VAL(tmp$)
             indx2% = indx2% +1
        WEND
    NEXT
    CLOSE #1
END SUB

'  This subroutine writes the array as a Cricket graph™ Fast
format file
SUB Save.Cricket(array(2),array.title$(1),N.col%,file.name$)
STATIC
    watch% = 4
    N.pts% = UBOUND(array,2)
    changecursor watch%
        OPEN file.name$ FOR OUTPUT AS #1

    tmp% = N.col%+1
    tmp% = INT(tmp%/16)
    PRINT #1,CHR$(tmp%);
    tmp% = N.col%-16*tmp%+1
    PRINT  #1,CHR$(tmp%);

    FOR indx% = Ø TO N.col%
        IF LEN(array.title$(indx%)) = Ø THEN
array.title$(indx%) = " ? "
        IF LEN(array.title$(indx%)) < 15 THEN
            PRINT  #1,CHR$(LEN(array.title$(indx%)));
            PRINT  #1,array.title$(indx%);
            FOR indx2% = LEN(array.title$(indx%))+1 TO 14
                PRINT  #1,CHR$(Ø);
            NEXT
            PRINT  #1,CHR$(1);
        ELSE
            PRINT  #1,CHR$(15);
            PRINT  #1,LEFT$(array.title$(indx%),15);
        END IF
        tmp = 12*(N.pts%+1)
        tmp1 = INT(tmp/256)
        PRINT  #1,CHR$(tmp1);
        tmp1 = tmp-256*tmp1
        PRINT  #1,CHR$(tmp1);
        FOR indx2% = Ø TO N.pts%
            PRINT  #1,USING"##.####^^^^";CSNG(array(indx%,
             indx2%))
        NEXT
    NEXT
CLOSE #1


    NAME file.name$ AS file.name$ , "STWK"
    setCreate file.name$ , "CGRF"
END SUB

 SUB errormsg ( msg$ ) STATIC
    win = WINDOW(Ø)
    PRINT CHR$(7)
    IF win = 4 THEN
        PRINT" THERE ARE TOO MANY OUTPUT WINDOWS OPEN TO OUPUT
ERROR"
      RESET
      MENU RESET
       END
   END IF
            WINDOW win+1,"ERROR WINDOW", (10,100)-(450,210),-4
            PRINT msg$
            default 5,"OK",50,50,80,80
             WHILE DIALOG(Ø) <> 1 AND DIALOG(Ø) <> 6 : WEND

             BUTTON CLOSE 5
         WINDOW CLOSE ( win+1 )
            END SUB

     '  This subroutine save the Array as a Cricket Graph
 Delimited Format
     '  This is old Code I used to use this years ago
 SUB save.Cricket.tab(array(),array.title$(),N.col%,file.name$)
 STATIC
        watch% = 4
        ttab$ = CHR$(9)
        N.pts% = UBOUND(array,2)
        OPEN file.name$ FOR OUTPUT AS #1
        PRINT #1,"*"
        FOR indx% = Ø TO N.col%-1
            PRINT #1,  array.title$(indx%);ttab$;
        NEXT
        PRINT #1,  array.title$(N.col%)
           FOR rindx% = Ø TO N.pts%
           FOR cindx% = Ø TO N.col% -1
           PRINT  #1,USING"##.####^^^^";array(cindx%, rindx%);
           PRINT  #1,ttab$;
           NEXT
           PRINT  #1,USING"##.####^^^^";array(N.col%, rindx%)
           NEXT
```


© The Definitive MacTutor, Vol. 4

```
        CLOSE #1
        NAME file.name$ AS file.name$,"CGTX"
        setCreate file.name$,"CGRF"
    END SUB

    ' this routine reads a Cricket Graph Tab delimited format
from the Disk
SUB open.cricket.tab(array(),array.title$(),N.col%,file.name$)
STATIC
        watch% = 4
        changecursor watch%
        N.pts% = UBOUND(array,2)
        ttab$ = CHR$(9)
        OPEN file.name$ FOR INPUT AS #1
        INPUT #1,tmp$
        IF tmp$ <> "*" THEN
            msg$ = "This file is not compatible with this
program"
            CALL errormsg(msg$)
            EXIT SUB
        END IF

        LINE INPUT #1,tmp$
        cindx% = 0
        tmp% = INSTR(tmp$,ttab$)
        WHILE tmp% > 0
            array.title$(cindx%) = LEFT$(tmp$,tmp%)
            tmp$ = RIGHT$(tmp$,LEN(tmp$)-tmp%)
            tmp% = INSTR(tmp$,ttab$)
            cindx% = cindx%+1
        WEND
        array.title$(cindx%) = tmp$
        N.col% = cindx%

        FOR rindx% = 0 TO N.pts%
            LINE INPUT #1,tmp$
            FOR cindx% = 0 TO N.col%-1
                tmp% = INSTR(tmp$,ttab$)
                array(cindx%,rindx%) =VAL( LEFT$(tmp$,tmp%))
                tmp$ = RIGHT$(tmp$,LEN(tmp$)-tmp%)
            NEXT
            array(N.col%,rindx%) = VAL(tmp$)
       NEXT

    END SUB
SUB default (BUTTONID%,Title$,Left%,Top%,Right%,Bottom%)
STATIC
  DIM R%(3)
  BUTTON BUTTONID%,1,Title$,(Left%,Top%)-(Right%,Bottom%),1

  setRect R%(0),Left%-3,Top%-3,Right%+3,Bottom%+3
  PENSIZE 2,2
  CALL FRAMEROUNDRECT(VARPTR(R%(0)),16,16)
  PENSIZE 1,1
  ERASE R%
  tmp = FRE(0)
 END SUB
```

Finding The Area Of A Region In C

James Plamondon
Albuquerque, NM

I received my MacTutor today, and was delighted with it, as always (even if my article on random numbers still languishes on the back burner). I was particularly interested in the article Regions with Aztec C in the C Workshop column [October '87, v3 n10 p27-32].

The article was concerned, in part, with estimating the area of a given Quickdraw region. The authors accomplished this with a function, "countpix()." Countpix() considered each pixel in the region's enclosing rectangle, and determined whether each fell inside or outside the region in question.

This, to use the authors' own words, is a brutal kludge. The authors intended to find the areas of regions which had been drawn freehand; accuracy can hardly be at a premium. Further, the authors state repeatedly that they are "estimating" the area of the region — so why work too hard at an estimation?

No "assembly-language optimization" will rescue an inefficient algorithm from inefficiency; it will just waste time faster. A more appropriate algorithm would be one based on a probabilistic estimation of the area of the region, based on a random sample of points in its enclosing rectangle. The number of points sampled could be very large (hundreds) and still be only a small percentage of the total number of pixels in a large region.

Consider a region bounded by a rectangle with its top-left vertex at (l, t) and its bottom-right vertex at (r, b). Assume a function (GetIntURand()) which would return a uniformly-distributed pseudo-random short integer in the range [l, r) or [t, b). We can re-write countpix() to return a 32-bit integer (a C long int), which gives an estimate of the area of the region in pixels:

```
long countpix(rh)
rgnHandle rh;
  {
    Point p;
    short i, t, l, b, r;
    long count;

    t = (*rh)->rgnbbox.a.top;
    l = (*rh)->rgnbbox.a.left;
    b = (*rh)->rgnbbox.a.bottom;
    r = (*rh)->rgnbbox.a.right;

    for (i = 0; i < NUM_PTS; i++)  {
        p.a.v = GetIntURand(t, b);
        p.a.h = GetIntURand(l, r);
        if (PtInRgn(&p, rh))
            count++;
      }
    return((count * (b - t) * (r - l)) / NUM_PTS);
  }
/* end of countpix() */
```

NUM_PTS will determine the accuracy of the estimation. It may be either a '#defined' constant or a variable, proportional to the area of the enclosing rectangle (or the length of one side). In both cases, the accuracy of the estimation will vary with the density of the region (ie., how fully it fills its enclosing rectangle). If NUM_PTS is a constant, the accuracy of the estimation will also vary with the size of the region.

This method is not as accurate as the method proposed by the authors. However, it is much more appropriate for their stated desire, it is much faster, and it, too, is amenable to assembly-language optimization.

XCMD's And XFCN's With Non-MPW

Steve Rotl
Portland, OR

In response to Danny Goodman's call for more information on writing XCMD's and XFCN's with non-MPW compilers (reported in Joel West's column, December '87), I am submitting my comments on using Consulair's MacC version 5.0 for this purpose.

First, grab the include file "HyperXCmd.h" from the disk

supplied with the HyperCard Technical Reference Package from APDA. This file contains the definition of the XCmdBlock structure used for passing information between HyperCard and the XCMD. The C source file must have a typedef to account for the type Boolean used in XCmdBlock. The following C code shows how to set up the source file.

```
#Options +Z
typedef char Boolean;
#include <HyperXCmd.h>
pascal void main(paramPtr)
    XCmdBlockPtr    paramPtr;
{
    /* body of XCMD */
}
```

The only real surprise here is that the function name must be "main" rather than the actual name of the XCMD (otherwise the linker will moan about not finding the label "main"). The other difference is that I was not able to get the include file "XCmdGlue.inc.c" to compile in MacC, so I don't include it at the end of the source file. The callback routines should be able to be called using the same approach as in Pascal (procedure DoJsr), but I haven't had the need to call any of them yet since the input parameters and return value can be accessed directly from the XCmdBlock structure.

The following Link Control File shows how to link the XCMD.

```
/Output XCMD
/Type 'RSRC'
XCMD
Standard Library
/End
```

The name of the compiled file containing the XCMD must precede the Standard Library file so that the XCMD code will be the first thing in the resulting segment. After compiling and linking, fire up ResEdit and open the linked file, which will contain three CODE resources. Open the CODE resource with ID=1, select from address 00000004 to the end (this will eliminate the four header bytes from the segment), and copy the selection to the clipboard. Close the linked file and open the stack that will contain the XCMD. Create a new XCMD resource and paste the code from the clipboard into it. Choose the menu item Get Info for the new XCMD, type in the exact name of the XCMD, and check the "Purgeable" box. Save the changes to the stack and quit ResEdit.

Finally, a working XCMD! (You did code it correctly didn't you?) Note: everything said here applies to XFCN's as well.

New Version Of Shift Mod Patch

Andy Voelker
Long Beach, CA

Here is a new version of the Shift Mod Patch that you published in the September, 1987, issue. This version will work with Desk Accessories; the previous version did not.

The ShiftMod Patch in the September, 1987, issue of MacTutor didn't work properly for desk accessories. Here is a patch to the original code that will fix the problem. It puts a prologue patch on _SystemEvent that uses the same code as our _GetNextEvent patch to modify keyDown and autoKey events if both the Caps Lock and Shift Key are held down at the same time (in which case it will return a lower case letter).

This modification to the patch is in two parts. The new code is surrounded by comments of asterisks. I have given a line or two of the original code on either side of the modifications so you can find where to put the modifications in the original code.

First modification: during the installation part that patches _GetNextEvent. Add some code to patch _SystemEvent just before the call to _BlockMove:

```
/* set the trap address to the space we
 * just got in the system heap. */
        move    #GetNextEvent,D0
        _SetTrapAddress

/*********************************/
#define  SystemEvent       0xA9B2

/* set up the address of the SystemEvent
 * JMP instruction */
        move    #SystemEvent,D0
        _GetTrapAddress
        lea     @sysEventAddr,A1
        move.l  A0,(A1)

/* patch SystemEvent */
        move    #SystemEvent,D0
/* calculate the after-move address of
 * the _SystemEvent patch */
        lea     @newSysEvent,A0
        lea     @first,A1
        suba.l  A1,A0
        adda.l  (SP),A0
        _SetTrapAddress
/*********************************/

/* now move it into place */
        lea     @first,A0
        move.l  (SP)+,A1
        move.l  D3,D0
        _BlockMove

Second modification: the patch to _SystemEvent. Put it after the
patch to _GetNextEvent, but before the label @last.

@eventPtrdc.l 0

/*********************************/
#define  RTS  0x4E75

@newSysEvent
        lea     @patchExit,A0
        move    #RTS,(A0)
        bsr     @tailPatch
        lea     @patchExit,A0
        move    #JMP,(A0)
        dc      JMP
@sysEventAddr
        nopb ~
        nop
/*********************************/
@last
```

Volume 4 Number 5

Multitasking Homework

Mathew Snyder
South Bend, IN

I am an undergrad at Notre Dame. I am invoved in a

research project in which we are constructing a communication facility similar to TCP/IP. For this project. I have been required to port a multitasking "operating system" (really an operating system enhancement) to the Mac.

The operating system kernel and the processes which use it run in the background. The technique I use to make all the code for the system remain resident is a little underhanded, and I would like to know if there is an easier way it could have been done. The current technique involves moving the code in the application heap into the system heap.

The tricky part which is most underhanded, is convincing the Mac's memory manager that the space now at the top of the system heap should be considered an allocated and locked block. I do it by directly manipulating the data base of the memory manager.

I considered allocating a block in the system heap and loading in the code I want to stay resident by hand, but the code is compiled and linked using an ordinary language system, and I didn't want to mess with allocating stack space, global variable space, resolving references, building a jump table, etc.

Is there a simple way to instruct the segment loader to fix things such that an application gets loaded into the system heap? (I suspect not.)

I am aware that it is possible to have your resourses loaded into the system heap. What would happen if I loaded the resources of interest using the resource manager? Would references be resolved, etc.? *[Messing around with the system heap is a Macintosh no-no. You might consider Lightspeed Pascal and the Mach2 Forth system. Both of these products use a kind of application specific multitasking that manages to co-exist with the Mac OS, but not without a lot of work and bug fixing. The Resource Manager cannot resolve code references in seperate code resources. It is not a linker. -Ed]*

Scanning for specific keys
Warren P. Michelsen
Page, AZ

Just wanted to drop a quick note your way in response to a letter in the March issue regarding scanning for a keypress. I had the same problem about a year ago and experimented enough with LS Pascal to satisfy my needs.

I'm just a little ashamed to say that I did not document everthing I found out regarding how LSP treats a keyMap and I don't have the time right now to reverse engineer the enclosed source code. Once I figured out how it works I thought I'd never forget.

Obviously, my routine(s) could be modified to accept a passed parameter to check for any key. I didn't happen to need that feature at the time. In fact, I've never had the need to check more than the option key. Oh, well. I hope someone finds it useful. *[Note that this unit does not handle all combinations of keys if both the shift and command key are held down together. As presented here, it correctly detects a single function key held down in conjunction with a normal key. -Ed]*

```
UNIT ModKeys;

INTERFACE

   VAR
      MyKeys : KeyMap;

   FUNCTION CommandIsDown : Boolean;
{ check the Command key to see if pressed . }
   FUNCTION OptionIsDown : Boolean;
{ Check the Option key to see if pressed .}
   FUNCTION CapsIsDown : Boolean;
{ Check the Caps Lock key to see if pressed . }
   FUNCTION ShiftIsDown : Boolean;
{ Check the Shift key to see if pressed . }

IMPLEMENTATION
{————————————————————}
   FUNCTION CommandIsDown;
   BEGIN
      CommandIsDown := False;
      GetKeys(MyKeys);
      IF BitTst(@MyKeys[1], 16) THEN {16 = Command key}
         CommandIsDown := True;
   END;
{————————————————————}
   FUNCTION OptionIsDown;
   BEGIN
      OptionIsDown := False;
      GetKeys(MyKeys);
      IF BitTst(@MyKeys[1], 29) THEN {29 = Option key}
         OptionIsDown := True;
   END;
{————————————————————}
   FUNCTION CapsIsDown;
   BEGIN
      CapsIsDown := False;
      GetKeys(MyKeys);
      IF BitTst(@MyKeys[1], 30) THEN {30 = CapsLock key}
         CapsIsDown := True;
   END;
{————————————————————}
   FUNCTION ShiftIsDown;
   BEGIN
      ShiftIsDown := False;
      GetKeys(MyKeys);
      IF BitTst(@MyKeys[1], 31) THEN {31 = Shift key}
         ShiftIsDown := True;
   END;
{————————————————————}
END.  { of unit }
```

Background Load Task
Erny Tontlinger
Steinfort, Luxemburg

Here are some comments about the article "A Background Task for Measuring Load" from Peter Korn in the C Workshop part of MacTutor, March 1988. I found the article and the program very interesting because I wanted to become more familiar with MultiFinder programming. But there was one bug and two details where I cannot understand why they were programmed like that (especially from someone working for Apple Computer).

In the function doLoad(), the current GrafPort must be saved, the GrafPort of the loadWindow must be set before drawing the bar graph, then the saved GrafPort must be restored. Otherwise the program will bomb if you try to open a disk accessory while the program is in the front. *[This is not true. The program runs just fine with desk accessories, which are taken care of while under MultiFinder in the DA layer. If you are not under MultiFinder, it is possible that some flakey DA's might*

cause a problem, but the program doesn't execute unless MultiFinder is active. -Ed]

Why that discussion about coding an absolute filename in the program? The program line in question (OpenResFile("\pGLA appl");) is completely superfluous because the resource fork of the application is opened automatically when the program is launched! *[See Peter Korn's response to this question below. -Ed]*

The WaitNexxtEvent() function can be coded much simpler in LightSpeedC (I'm using version 2.11) as follows:

```
#define WaitNextEvent(eventMask, theEvent, timeOut, mouse-
Box)\
    _WaitNextEvent((eventMask), (theEvent), (timeOut),
(mouseBox))
Pascal short_WaitNextEvent()=0xA860;
```

Notes on Background Load Task

Peter Korn
Berkeley, CA

I was very happy to see my article, System Load Under MultiFinder, published. Having seen and learned from the many fine articles in your publication I was very happy to make a contribution of my own. In fact, I'm working on a few other subjects (XCMDs for HyperCard, etc.) that I hope to turn into articles in the near future.

I was saddened to see the article intimating that I am an employee of Apple Computer. While it is true I work there full time, I do so as a contractor, which isn't quite the same thing as an employee. Though this distinction isn't terribly important one (the content of the article is just as valid either way), I would like to set the record straight.

While we're on the subject of record-straight setting, I should also mention that the current version of UW (Unix Windows, by John Brunner), version 4.2, now handles UpdateEvents purely as UpdateEvents, and doesn't do anything else with them. In fact, UW 4.2 is fully MultiFinder friendly to the best of my knowledge.

One bit of code that perhaps should have been in GLA but didn't make it into the copy I sent you is a test for MultiFinder's "Temporary Memory Calls". While this isn't strictly necessary when running under System 4.2 (where, if WNE() is implemented, so is MFMaxMem() and its associated calls), it may become a problem if Apple ever seperates the two (by, say, putting WNE() into the System proper, but leaving MFMaxMem() and its bretheren as part of MultiFinder only). In that case, GLA would run even with no MultiFinder present, but crash the moment MFMaxMem() is called. Probably the best way to implement this would be to set a boolean to true if the Temporary Memory calls were available, and then simply disable the "Compress Juggler RAM" menuItem if that boolean were false:

```
#define   JugglDispatch 0x8F
#define   UnImplTrapNum 0x9F
#define   SwitchPtr (long *)(0x282)

boolean   tempRAMhere;

   tempRAMhere = (((*SwitchPtr == 0) || (*SwitchPtr == -1))
&& (theWorld.machineType >= 0 ) &&
(NGetTrapAddress(JugglDispatch, ToolTrap) !=
NGetTrapAddress(UnImplTrapNum,ToolTrap)));
          o
          o
          o
   if (!tempRAMhere)
      DisableItem(configureMenu, 3);
```

Lastly, your comments on page 41 "*...Peter does another thing we don't recommend: he opens a specific resource file by file name...*" is made a little hastily. In LightSpeed C, one normally puts in the line:

```
"OpenResFile("\pResourcesFileName.rsrc")",
```

which is a queue to Lightspeed C to go and look for the resources in that file when LightSpeed C launches the program directly from itself. This is also a way of informing LightSpeed C where it should go to find the resource file it will use when it links the file. This line is in there for development purposes, and is normally ignored when the program is compiled and run (so sayeth the LightSpeed C manual). In fact, the original source code that I sent you contained the line:

```
"OpenResFile("\pLoad Average.rsrc")",
```

and not

```
"OpenResFile("\pGLA appl")",
```

which was how you apparently edited it to be before printing. Had you left it as I sent it to you, you would have probably realized that it couldn't have been a hard coded name, as the program's name is "GLA" and not "Load Average.rsrc". Apparently in the rush to publish the March issue you missed this detail. *[I was unaware LS C works in this manner. They should change it to work like their LS Pascal product, which uses a dialog box rather than a code hack to deal with resources. Sorry. -Ed]*

Folder Heirarchies & User Interface

David Dunham
Goleta, CA

Interesting to see the user interface letters in MacTutor. The interface is an all-too-often neglected part about writing a program. A lot of people have ideas that I think are impractical (of course, the only real way to tell is to try them with real people, in real situations). For example, Ted Johnson's idea for folder hierarchies which run up sounds nice at first, but it means menus work two different ways —normally you click on the menu and drag down. Now you'd have some menus that you drag up to operate. (Yes, pop-up menus as implemented in the newish PopupMenuSelect() trap also violate the one-way-to-use-a-menu principle. Does this suggest they should be avoided whenever possible?) But this is a philisophical point, and I could be wrong. Less subjectively, the item appears at the top of the dialog. There is a lot more space for a long hierarchy if the list runs from top to bottom, instead of bottom to top (it could easily bump into the menubar or top of the screen). Perhaps the pop-up should have been put under the scrolling list, but that would probably have conflicted with existing dialogs. If Apple had designed the original Mac with HFS in mind (from the begin-

ning I wondered why they didn't), Johnson's idea would be good. With the existing Mac, it's not compatable with the common way of doing things, and its advantages don't seem to outweigh this disadvantage.

How to Animate a Watch Cursor
The Midnight Hacker
Atlanta, GA

Responding to some comments made in the March Mouse-hole Report...

Regarding Consulair's 1) lack of technical support, 2) less than a full Toolbox (support for IM V. 5 calls), and 3) Mac C Jr.; Consulair Mac C has had one foot in the grave for over a year now. Mac C Jr. is the original Mac C without library sources or technical support. Competition from Lightspeed C and MPW have eaten Consulair's lunch, so it is most unlikely the product will ever be upgraded again.

Mac C Jr. is a last gasp attempt to squeeze $$$ off the bottom of the market (as the original Mac C, the first C development system for the Mac (and very pricey at over $400), was meant to cream $$$ off the top). Ditto Consulair 68000 assembler. Simple economics: the end of a product life cycle due to overwhelming competition. Mac C is a solid product, but LightSpeed is THE Mac Code Yuppie compiler, and MPW has Apple's name on it.

Speaking of Lightspeed C... Version 3.0 is due in April-May. Anyone who bought v2.15 after February 1 recieves a free upgrade. The new version has a source-level debugger (ala LS Pascal), bug fixes, improved editor, and much improved documentation.

FullWrite Professional may be a memory hog (it requires a Meg for itself to run), but it is going to give Word and PageMaker both a serious run for the money. FullWrite is one amazing piece of software. The only shame is the early advertisements that made it look like a vaporware hype (which it was at the time). The boys of Ann Arbor (er, Ashton-Tate) made the tough but correct decision not to release FullWrite until it was done (remember Word 3.00?!). Better to eat crow up front than have the magazine reviewers sink your product once it hits the streets.

Anamated cursers. Do it yourself. Keep track of what cursor is active using a global (curr_cursor below).

```
#define WATCH1  ?  /*cursor resource number of 9 o'clock */
/* WATCHes are consecutively numbered cursor resources */
#define WATCH8  ? /* cursor resource number of 9:50 watch */
#define NEXTWATCH 10 /* ticks between watch changes */
#define EVENTFLUSH 0xFFFFFFFF

(global)
long watchtick;   /*the last TickCount the watch was
changed*/
short curr_cursor;     /*current cursor (resource number)*/
CursHandle watch[8];  /*handle to watches*/

(local)
long now          /*be here..*/

if (curr_cursor >= WATCH1)
    {
    FlushEvents(EVENTFLUSH,NIL); /* no user event build-up */
    now = TickCount();  /*check if it's time to change
watch*/
    if((now -watchtick)>=NEXTWATCH)
        {
        watchtick = now;       /*new watch indicator*/
        /*set the cursor to the next watch*/
        if(curr_cursor == WATCH8)  curr_cursor = WATCH1;
        else                       curr_cursor++;

        /*show the appropriate watch cursor*/
        SetCursor(*(watch[curr_cursor-WATCH1]));
        }
    }
```

It's laughable that someone wants Apple to animate their cursor for them. I guess the Magic Toolbox has made us spoiled children. If you need an attitude adjustment, try programming an MS-DOS boat anchor. That'll animate your cursor!

Microsoft news: the DOS kids are beavering on their CD-ROM Reference Set DA for the Mac (similar to the Bookshelf Reference CD-ROM currently available for everyone's favorite boat anchor). It'll make a great Christmas Gift (if the schedule doesn't slip).

MultiFinder Bug in the Background?
Larry Schmitz
Athens, GA

Thank you for your article on the MultiFinder friendly Draw Plotter (Feb. 88). It was most useful. There were, however, a number of corrections that needed to be made to get the program to run. The procedure, "doMessage", was missing and the about box and the PICT it contained required some redoing. *[The doMessage proc was indeed left out, but printed in the March issue. The code and about Box compile and run as printed. -Ed]*

One point that the program did not address was doing background processing. I have written a program that runs in the background under MultFinder. However, it has one problem that I have had to write a clumsy kludge to get around. This problem is also exhibited by your program if you set the Can-Background bit in the size -1 resource. Perhaps you or one of your readers can shed some light on this topic.

The problem only occurs if WaitNextEvent is called, the CanBackgroundbit is set and the application does not have a window open. In order to demonstrate the problem using your MiltiFinder Friendly Plotter program as an example, do the following. First, compile your program (I used LSP 1.11 and make the corrections mentioned in the first paragraph). Then use ResEdit to set the CanBackground bit. Next, run MultiFinder and launch the Plotter program. Make sure that none of the windows from the Plotter program are open. Open a desk accessory and then close it so that the Plotter program again becomes the front most layer. At this point the Mac will hang and must be restarted. In order to get around the problem, I call SystemTask/GetNextEvent rather than WaitNextEvent when FrontWindow returns **nil**. Unfortunately, this is not very friendly to other background tasks. Whose bug is this? *[The plot program does indeed hang the Mac under the conditions you describe, and the problem does go away if you check for a front window before calling WaitNextEvent. There must be some*

connection between accepting null events as a background task, the desk acc. layer manager, and WaitNextEvent expecting to find an active window. I can't imagine what that connection is, but it seems like either a MultiFinder bug or a bug with the Desk Acc. handling under MultiFinder. Peter Korn and I spent an hour looking into this and we did confirm that his graphic load background task shows the same problem, if you remove the showWindow command. So there is a high probability this is a MultiFinder bug. -Ed]

Floating Windows in C

Vincent D. McGarry
Austin, TX

In answer to the questions posed by Kenrick Chan and Ajay Nath, I have been using the follow code (Aztec C) to "float" windows. Calls to DragWindow() cannot be used since it calls SelectWindow(). Instead I do some necessary preparation and then call DragGrayRRgn() and MoveWindow() directly. SelectWindow() is never called. The window I want to float is the Ghost Window. I have not tried to do this with GrowWindow(). All the windows in my application are "noGrowDocPro"s and therefore act like the tools palette in FullPaint. I have not done this with more than one window "floating" but soon will be since my application requires that the active window, being quite large on the screen, be behind the several other comparatively small windows that accompany it.

```
*define       GHOST       0xA84
*define       BADMOVE     0X80008000

setupwindows()
{
  WindowPtr  *GhostWindow; /* and other declarations */
  GhostWindow =GHOST
  FloatWindow =GetNewWindow(FloatResID,&FloatRec,-1L);
  *GhostWindow= FloatWindow;
  ShowWindow(FloatWindow);
  BringToFront(FloatWindow);

  /* rest of setupwindows      */
}

mousedown()
{
  /* local declarations and other processing     */

  case inDrag:
    Screen = screenBits.bounds;
    SetRect(&limitR, 4, 24, Screen.right - 4, Screen.bottom -
4);

setRect(&slopR,  Screen.left,  Screen.top  +24,  Screen.right,
Sceen.bottom);
    local_mouse = global_mouse;
    GlobalToLocal(&local_mouse );
    GetWMgrPort( & wm_port );
    windstruck =  NewRgn();
    CopyRgn(((WindowPeek)whichWindow)->strucRgn, windstruct);
    GetPort(&saveport);
    SetPort(wm_port);
    ClipRect(&wm_port->portRect);
    newloc = DragGrayRgn ( windstruct,  /*new window loc*/
                           pass(global_mouse),
                           &limitR,
                           &slopr,
                           0,
                           0L   );
    DisposeRgn (windstruct);
    if ( newloc != BadMove)
        MoveWindow( whichWindow,
            LoWord(newloc)+global_mouse.h - local_mouse.h,
            HiWord(newloc)+global_mouse.v - local_mouse.v,
            FALSE);

    SetPort(savePort);
    InvalRect (&savePort->portRect);
    break;
  /*        other processing       */
}
```

Floating Windows the Simple Way

Ian McLellan
Waterloo, Canada

Here is a way to get a certain window to always stay in front of another window. What I tried is pretty sleazy, but it works. In my main loop, I put:

```
IF FrontWindow <> theWindowToStayInFront THEN
    BringToFront(theWindowToStayInFront);
```

It seems to keep the "behind" window active, yet puts the window (usually a menu window) on top of it. This probably breaks a couple of Apple's laws, and I haven't tested it too extensively, but I couldn't think of another way to do it.

Volume 4 Number 6

[TML has returned to the Mac market with a new version of TML Pascal that runs under MPW and supports objects and MacApp. This is the first non-APDA product to be available from a third party that supports MacApp. This is important because it means more widespread availability of MacApp compatible MPW tools than just through APDA, which enjoys an Apple supported monopoly on distribution. Here is an advance look at TML Pascal version 3.0, being announced this month at Mac Hack in Ann Arbor, Michigan. -Ed]

TML Pascal v3.0

Tom Leonard, TML
Jacksonville, Fl.

Version 3.0 of the TML Pascal compiler will be a completely new implementation of TML Pascal. The most significant change for this product is that it will be converted to operate with the Macintosh Programmer's Workshop (MPW) as an MPW Tool.

The following is a list of the major features provided by TML Pascal v3.0:

- Operates within the Macintosh Programmer's Workshop
- Complete support for Object Pascal
- Compiles MacApp
- Native 68020 and 68881 code generation
- Conditional compilation
- Assembly source code output as well as object code output
- Constant expression evaluation in declarations
- Performance analyzer
- Complete integration with other MPW languages including MPW C and Assembler.

In addition to the Pascal compiler, the product will also ship

with a Pascal pretty printer (TMLPasMat) and Pascal cross reference tool (TMLPasRef).

TML Pascal will support the development of any type Macintosh program. These include:

- Desktop applications
- "Textbook" applications
- MPW Tool applications
- Desk accessories
- Definition procedures (WDEF, MDEF, CDEF, LDEF, etc.)
- FKEY resources
- INIT resources
- HyperCard XCMDs and XFCNs

and more...

Examples of how to create all of these various types of programs are provided with either the TML Pascal product or the TML Source Code Library.

In addition, TML Pascal will provide several MPW "scripts" which will implement custom menus and other user interface elements to make using the MPW shell easy for the beginning programmer. Compiling and executing a program will be as simple as selecting the "Compile & Run" command from the "TML Pascal" menu. In fact, many users may never find it necessary to use other MPW commands to compile and create programs.

External Tool Specification

Syntax TMLPascal [*option...*] [*file...*]

Description Pascal language compiler.

Input: One or more Pascal program or unit source code text files. By convention, Pascal source file names end with the suffix ".p"

Output: The compiler can create two types of output: MPW object code (the default) or MPW assembly source code. The output is written to a file whose name is the input source file name with the ".o" or ".a" suffix respectively.

There is no output to *standard output*.

Diagnostics: Compile time errors are written to *diagnostic output*. Progress and summary information is also written to diagnostic output if requested.

Status: The following status values are returned to the Shell:

0 Successful completion
1 Error in parameters
2 Compilation halted

Options:

-align Align all data items on longword boundaries.

-asm Compile the source file to assembly source output rather than object code.

-d *name*= TRUE | FALSE

Set the compile time variable *name* to TRUE or FALSE

-i *pathname* [,*pathname*]...

Search for include or USES files in the specified directories. Multiple -i options may be specified. At most 15 directories will be searched.

-mc68020 Generate code to take advantage of the 68020 processor.

-mc68881 Generate code to take advantage of the 68881 co-processor.

-o *outName*

Specify the pathname for the generated output file (object or assembly). If *outName* ends with a colon (:), it indicates a directory for the output file, whose name is then formed by the normal rules. If *outName* does not end with a colon, the output file is written to the file named *outName*.

-ov Turn on overflow checking.

-p Output compiler version and progress information to *diagnostic output*.

-r Suppress range checking.

-t Report compilation time to *diagnostic output*.

-u Initialize local and global data to the value $7267. Used for debugging.

-z Turn off the output of embedded procedure names in the object code. This option is equivalent to the {$D-} compiler directive. By default, the compiler creates the embedded procedure names for use by TMON, MacsBug, and other debuggers.

Language

TML Pascal v3.0 will be a compatible ANS Pascal and MPW Pascal compiler. In addition, a significant amount of time will also be spent further improving the code quality of generated code. Because the current version of TML Pascal is already very much compatible with MPW Pascal, no major changes are required. The following is a list of the more significant changes to be made for TML Pascal v3.0. See the TML Pascal v3.0 Language Reference Manual for a complete definition of the language.

- Conditional compilation
- Short circuit boolean operators (&, |)
- Exponentiation operator (**)
- Identifiers may begin with an underscore
- Implement the "C" subprogram directive
- Fully implement Object types
- Type REAL changed to be equivalent to SINGLE
- Support 96 bit extended reals for 68881
- Support constant expressions where constant literals are normally expected
- Support sets in the range 0..2039
- Untyped files
- Type casting between ordinal types of different sizes

The Trouble With FONDs

Cliff Joyce
Northridge, CA

By the way, I noticed the discussion in a recent MacTutor issue regarding the problems with bitmapped fonts behaving strangely in applications that use fractional character widths.

The problem goes back to older versions of the Font/DA Mover Which improperly set bit 14 of the ffFlags word when building FOND resources from scratch. This flipped bit tells the application that the FOND contains a fractional width table. The application then uses ffWTabOff to find the address of the width table. But since ffWTabOff is zero in this case, the application

points to the beginning of the FOND—usually causing characters to overwrite themselves when being drawn.

We have removed all FONDs from our World Class Fonts collections, and included Font/DA Mover v3.6, which properly sets bit 14 of the ffFlags word.

However, a safer solution might be for programmers to make two tests to determine if a FOND contains a fractional width table. First test the bitflag, then test ffWTabOff. If ffWTabOff=0, assume there is no fractional character width table.

By the way, *Font/DA Mover v3.6 now improperly sets bit 15 of the ffFlags word* when creating generic FOND resources (unless my Inside Macintosh volume IV is not accurate in describing the setting of bit 15). This may be a moot point as I am unaware of any applications that use this bit for anything (yet).

Oh, yes—enclosed is my check for another year's subscription. That's it for now. Keep up the good work...

Volume 4 Number 7

Announcing...
Steffanie Sheets

Steve and Patricia Sheets are very, very pleased to announce the birth of their daughter, Steffanie Virginia Sheets. Steffanie was born Monday May 30, 1988 at 11:55 AM. The blond hair, blue eyed girl weighed 6 lbs., 12 oz at birth, while the labor lasted just under 6 hours. The mother, the daughter and the panic stricken father are enjoying each others company for the next 2 weeks at home.

Hope to talk to you all when I get back! *[Steve Sheets is a contributing editor of MacTutor and responsible for nearly all the articles on color and the Mac II in the last year. We wish him and his family well. -Ed]*

Credit For Tear-Off Menus
Mike Bolch
President, Radius Inc.

We were flattered by all of the references to the Tear-off Menu feature of our Radius Two Page Display in your April, 1988 C Workshop. This feature is indeed both flashy and useful, and greatly appreciated by our customers.

The primary author of the software, however, is Ted Cohn, of the Radius engineering team. I'm sure Ted would be the first to acknowledge Andy's advice and support, but I'm equally sure that he was chagrined to see all of his work attributed to somebody else. I know I'm always upset when people confuse me with Tom Selleck...

$50 MatLab & FaceIt Revealed
Dan Kampmeier, FaceWare
Urbana, IL

As a developer of a Macintosh programming tool (FaceIt – see this or a recent issue of MacTutor for ad), I naturally keep a close eye on related products and reviews in major Macintosh magazines. Unfortunately, many reviewers seem to have had limited exposure to the wide range of available Mac programming tools, and the resulting reviews are full of "Mac-hype".

One of the most hype-ridden reviews of recent date was that appearing in the April 26 edition of MacWeek. The 1-page review entitled "Prototyper: Build-it-yourself software" might have been more appropriately entitled "Proto-Hype". When the reviewer finally did address potential problems with the product, he abruptly ended the discussion by saying that a call to the developers revealed that "all our wishes had already been anticipated". I called MacWeek to find out who this reviewer was (maybe we could get him to do an equally naive review of FaceIt!). Their response was that they really didn't know much about the review (they read it while we talked).

What's bad about such reviews is that they ultimately do a disservice to both the developers and the users (in this case programmers). In the above case, for the example, Prototyper from SmethersBarnes *is* a great product for *certain types or phases of development.* The reviewer should have made it clear under what circumstances the use of the Prototyper is warranted and what its limitations are. In our opinion, it is best suited for the first 10-30% of program development where one is creating a basic event loop, menu handling, and window resources. It can generate source code (currently limited to Pascal) to which the rest of the program code is added (the 70-90% of a program that *does* something). [Can't you just see those Apple reps carrying around a copy of Prototyper to demonstrate that all you have to do to write programs on a Mac is point and click! Get real.]

Thus Prototyper, unlike many other programming tools does not relieve you from having to dig through hundreds or thousands of lines of interface-related source code, it just helps you get *started* creating such code. A call to SmethersBarnes not only confirmed this overall impression of the limitations of Prototyper (they are more open about this on the phone than in their advertising), but also resulted in their disclosing a revealing bit of information: one of the companies using Prototyper asked for a special version which did *not* generate source code since they already had programmers who could do that! Thus it was the non–programmers who were being given Prototyper as a means to let the programmers know what they wanted in an interface. This conflicts with the reviewer's claim that Prototyper was "designed primarily for developers".

This brings us to MacTutor. Although generally a happy camper, the recent review of MatLab from Math Works by Paul Snively (both from Massachusetts) in the May 1988 issue of MacTutor caught my eye. This product retails for $895, but there is a competing product called "MacMatLab", based on the same original Fortran program which retails for just $50 [yes, that's an $845 difference]. In fact, after reading this review, I find very little difference between these products, although no mention of MacMatLab is made by Paul in spite of the fact that it has been advertised in MacTutor! It is hard to believe that Paul was unaware of the existence of MacMatLab. Anyway, before you run out and spend a whopping $895 for MatLab, you might want to try a $50 version of a very, very similar-functioning program!

Which raises an interesting question: Why can QED Associates charge just $50 for their version of MatLab? The answer is that they used our product, **FaceIt** ($50-$100), to add the Mac interface to an existing program. FaceIt is a collection of preformed resources which can be used by programs created with

almost any language and compiler. It "instantly" adds a Mac interface that handles such things as editing of text and pictures, file handling, and printing. One copy on disk can even be used by multiple programs at the same time. It is well-suited for the conversion of existing programs for use on the Mac, and is also useful in cases where a programmer does not have the time or resources to write every little program he/she creates. In July a version will be released which includes "instant" spreadsheet-like windows which can be linked to any array in memory.

So, contrary to what is often heard in MacTutor, there is a fast way to add a Mac interface: FaceIt. Moreover, our sample programs cover text editing, plotting, drawing, animation, and background processing. The current $100 version includes sample programs for 10 or more different compilers on 5 disks (or $50 for one compiler on 1 disk). FaceIt is not for everyone, but if it fits your needs then we are confident that there is no better value when it comes to development tools. The biggest criticism we get from users is that it should be priced higher so we can afford fancier advertising and increase its perceived worth. [Translation: Buyers are ignorant, so run only full-page color ads full of hype, don't be afraid to charge a ridiculous price, and most importantly, always subtract 5 cents from the price before putting it in print.]

Conclusion: Someone needs to take a careful look at the range of Mac developers tools and compare them with respect to price, ease-of-use, and functionality. There's just too much hype out there for the average programmer to see through. *[Actually, the opposite may be true: there is not nearly enough information out there for developers to know what is available. -Ed]*

Volume 4 Number 9

Mac Power Supply History Reviewed

Dr. Ray A. Gaskins
Hampden-Sydney, VA

It could be argued that 1987 was the year of the power supply (analog board) problem. Don Ritter, writing in MACazine, mentions the power supply in seven out of twelve of his M.U.G. WRESTLING columns. MacTutor, in six of its monthly issues, devotes more (and useful) words to it than any other publication. The Active Window (Boston Computer Society publication) mentions something about the power supply in three of its monthly issues. Perhaps not surprisingly, MacUser and MacWorld mention the power supply in only one issue each.

The earliest reference to the power board problem that I have seen is one mentioned by Ritter in MACazine (Jan 87, page 61). He references an article by Howard Upchurch which appeared in the July/August 1986 issue of Apple Gram. Upchurch blames the problem on two underrated capacitors and on the flyback transformer. Other 1986 articles referenced by Ritter are a "bad power supply board survey form" in which Apple admits to a power board problem with Mac Plus upgrades. However, MACazine itself makes no mention of power board problem in 1986 (nor, for that matter, do MacUser, MacTutor or MacWorld).

Apart from recommending the removal of the heavy aluminum RFI shield mounted across the top of the power supply board as a means of increasing air flow and reducing heat, Ritter has little to suggest short of suing Apple. Instead he tells an endless string of horror stories about multiple power board failures.

In my own fixed population of just over 100 Macintoshes, I have the full range of Macs (1984-1988). I can remember two Macs out of this population that seemed like characters out of Ritter's horror stories. In both cases, replacing the video tube fixed the problem. Therefore, my advice to anyone whose mac eats power boards (say, three boards in six months) would be to replace the video tube (along with the third or fourth power board). A symptom of this problem is a discoloration due to heat of the four-pin connector that connects the video tube to the J1 connector on the power board and a history of eating power boards.

I believe that there is some truth to the rumor that Apple felt that part of the power board problem was due to the procedure being used to discharge the video tube - you know, two crossed screw drivers. I have lost a couple of power boards because of this and began not discharging the video tube for doing routine things not involving the power board (e.g., replacing the logic board). My failure rate declined. Now there is a neat tool for discharging the video tube that meets Apple's approval and I use it religiously.

In the 15 months prior to July 1987, I replaced 13 power boards. In the 9 months since then, I have replaced only 4. I attribute this to three things: Loy Spurlock, Chuck Rusch, and Mysteray. MacTutor published long and detailed letters from Loy Spurlock (March 87, page 4) and Chuck Rusch (June 87, page 13) concerning the power board problem and what you could do about it (short of suing Apple). Mysteray (July 87, page 17) wrote two long comments in MacTutor's Mousehole Report concerning the J1 connector on the power board - why it tended to develop a cold solder joint, how to detect it and how to fix it. I am grateful to these three people for their words of wisdom.

Since July 1987, I have had 10 power board problems that, prior to reading Spurlock, et. al., would have meant 10 power board swaps. However, applying their advice, I was able to save 6 of these boards by resoldering. The symptoms were varied:

(a) three had the classic thin vertical white line in the center of the screen,

(b) two had the shakes (screen jitter), occasional spikes and expanding/contracting screen,

(c) three had horizontal lines across the top and/or bottom of the screen, and

(d) two had a faint vertical line just to the left of center.

Resoldering the four pins of the J1 connector fixed two (a)'s, one (b) and two (c)'s. Resoldering two other joints that appeared dull under close inspection fixed the other (b).

Resoldering had no effect on one of the (a)'s, one of the (c)'s nor on either of the two (d)'s. (Rumor has it that the faint vertical line means that the power supply will fail within six months.) Fixing six out of ten power boards by simple resoldering isn't bad. These Macs range from 128K to Mac Pluses. None had fans.

Using a jewelers eye piece (10X), I also examined the joints at J2 (9 pin connector) and J4 (11 pin connector) on each of these boards. More often than not, one or two solder joints on each end

showed cracks. Resoldering these, although good preventative maintenance, is usually not as critical as resoldering the four joints at J1.

I looked at a couple of the replacement power boards and noticed that some of the connections had been resoldered by hand, including the four pins of the J1 connector. I couldn't tell their resoldering from mine. The only difference was that they put on a new paper backing with new double-stick pads. If you are careful in peeling back the double-stick pad (use a plastic video alignment tool with a screw driver blade to help peel it back), you won't have much residue to clean off before resoldering and you can restick the pad without applying additional glue.

What caused the cold solder joints? I believe that the explanation given by Mysteray (loose video yoke connector) is probably correct. Therefore, I always tighten this connector whenever I resolder the pins at connector J1. For a thorough explanation of this, see Mysteray's comments in MacTutor (July 87, page 17).

As far as voiding your warranty is concerned, after 90 days you are on your own unless you have AppleCare. Since the connectors we are talking about are not heat sensitive components, there is very little likelihood of making matters worse by resoldering and there is a better than a 50% chance of fixing the problem. But, if you are still under warranty or if you have AppleCare, you don't have to worry about resoldering - just let your friendly dealer replace the power board.

One suggestion that Don Ritter probably made in jest turns out to be a good one. He suggests that you "buy yourself a smoke alarm and place it above your Mac." I'll go him one better. If you intend to leave your Mac on unattended, install a stand-alone automatic halon fire extinguisher as well as a smoke detector above your Mac.

HyperCard Needs a Diet
Neil Rieck
Kitchener, Ontario

I just sold my 4 year old Mac (it was a 512KE with an external floppy when we parted company) in order to buy an SE with an internal hard disk. I thought that a 20 Meg disk and one Meg of RAM would satisfy my computing requirements through 1990 until I loaded the HyperCard (version 1.0.1) package that accompanied the SE. I now believe that HyperCard is a scheme by Apple to sell hard disks and memory upgrades (see "What you need to use HyperCard" on page xvi of the "HyperCard User's Guide").

Although HyperCard seems very powerful, I can see no reason why the STACKS (HyperCard programs) must be so large. A very nasty example is Apple's "1987 HyperCard Supplement" which is 773.5K bytes (Data 762.3 K, Resource: 11.2K). An ASCII dump of the Data Fork revealed that the script commands are not stored as tokens (as was done in AppleSoft BASIC days when both RAM and Disk space were scarce), but are actually in their original text form. Didn't we learn anything in the 70's? I found 957 occurrences of "MouseUp" which would save 4785 bytes if replaced with 16 bit tokens (or 5742 bytes in the case of 8 bit tokens). Can you imagine the savings if ALL the script commands were tokenized and SPACES between them removed? (Note: The thought of jump tables and command tokens reminds me of the trap dispatcher).

At first glance, you would think that the 221 page "HyperCard User's Guide" would tell you everything you needed to know about HyperCard. Wrong! If you want to know how to SCRIPT (program in HyperCard) or even get a list of the legal script commands, you must order the "HyperCard Script Language Guide" from APDA. This would be similar to getting a BASIC language package, then finding out that you can only use it to run/modify the demos until you purchase further documentation. At least Apple could have listed the script commands in an appendix.

I am a VAX programmer by day (I am fluent in several languages) and was quite disturbed by comments from colleagues who criticized 4GL type packages like HyperCard. Although HyperCard seems to be a hardware hog as I've just shown (but it can be fixed), it does allow the average person to program the Mac without having to know about GrafPorts & GetNextEvent, etc.

The best analogy I can come up with is the automobile. The first cars required the owner to understand the basic theory of operation so he could crank start the engine, manually advance the spark, set the choke, and shift the gears. Today's cars do all these things for you which makes the car available to more people. This increases sales, which drops the price for everyone. Just imagine, a more powerful Mac at a cheaper price.

My advice to MacTutor readers is to accept and learn HyperTalk (the language of HyperCard) as another language because the script market will be huge, and there will be people who will find this method of programming difficult and will require your advice. A second point to consider is that there will be GOOD scripters and BAD scripters, and if MacTutor readers set a high standard now, we can have a positive effect on what we will be forced to deal with in the future. And if after all this there are some people who are still skeptical of newer and easier to use software packages, please remember that in the movie "2010", Doctor Chandra (the system designer) talked to HAL & SAL when it was convenient to do so, but also used a keyboard when it was required.

Notes on the Modifier Keys Article
Warren Michelsen
Page, AZ

I just wanted to correct the mis-impression you may have created in some people's minds regarding the short LSP unit source of mine which you published in the May issue. You may recall that it showed how to detect the press of a modifier key in LSP and was sent in response to a writer's Question in the March issue.

Most importantly, the functions, as published, do not require a posted event, that is, they will detect the press of a modifier key entirely independently of "a normal key", contrary to what you stated. While I suppose the functions *could* be used in lieu of checking the modifiers flags of an event, that is not what I wrote

or use them for. Typical uses are: Upon program start "if OptionIsDown then" go directly to a certain function (much the way Font/DA Mover "starts up" in DA mode of the option key is down at start. Or, "if OptionIsDown" when a user selects " About..." then I bypass the About dialog and go directly to the instructions, which are normally selected from a button in my About dialog.

Secondly, while the unit indeed "...does not handle all combinations of keys if both the shift and command key are held down together..." it is certainly easy enough to call multiple functions in succession: "if (OptionIsDown and ShiftIsDown) then..." It beats having to remember numerous constants representing each possible combination of modifier keys.

Your readers might appreciate clarification of these points.

Volume 4 Number 10

BNDL Sleuthing
Georg Newesely
Berlin, West Germany

I am a 17-year-old Macintosh admirer from Berlin, West Germany. Every day, I spend a lot of time with my Mac SE and your fabulous magazine. I have been working with Pascal for 1/2 year now. This year I started with assembly. At this time, I come to you with two questions.

(1) The Video Whiz reported in the Mousehole Report on how to use a UniDisk 3.5" for an Apple II on the Mac. Where can I get detailed information on how to hook it up (I don't have an external drive)?

(2) I have been writing a program called "Icons". It loads up the desktop file, investigates with string manipulations the bundle resources to find the ICN# IDs of the applications and of the files belonging to them. It scans the BNDL IDs using a loop:

```
for index := 1 to CountResources('BNDL') do
  begin
    theHandle := GetIndResource('BNDL', index);
    if theHandle <> NIL then...
  end;
```

This is complicated and does not allow the direct access to the IDs. Is there a way to get (a) the BNDL IDs directly without any scanning, and (b) the ICN# IDs without any string manipulation?

At this time, I want to thank you for many happy hours that I spent with my Mac & MacTutor™. No one else brings so many aristocratic articles in one convenient package as MacTutor™ does.

Prototyper Review
David V. Moffat
Raleigh, NC

I have to face the music: I wrote the Prototyper review that Dan Kampmeier complained about in this column (*Letters*, **MacTutor**, July, 1988.)

Let's face it: a non-technical magazine like **MacWEEK** needs a non-technical review, one that explores only the surface of a complex product.

Hype? Is describing the many powerful and easy to use facets of a program hype? That attitude flies in the face of what we like about the Macintosh. Mr. Kampmeier is being facetious.

Prototyper is a ground-breaking product, unlike any other. If Mr. Kampmeier's product is similar, but better, he must face the fact that it is *his* responsibility to get that fact across in his advertising. On the face of it, his product seems to be just like several other pre-written program shells and libraries.

In my defense – not just to save face – I must explain the circumstances behind that review. The review (and title) printed were not what I submitted to **MacWEEK**; the editor was faced with the problem of making a 1,400-word, 6-picture review into a 1,000-word, 1-picture review. He did a pretty good job.

In addition, Mr. Kampmeier's product had not yet shown its face when I wrote the review.

I also offered to do a *technical* review of **Prototyper** for **MacTutor**, but I had no response (sorry, deadlines and space considerations are at a premium right now - ed).

In any case, if anyone took my review at face value and bought **Prototyper**, he got his money's worth. I do not feel that I lost face by publishing it.

Another point: If Dan Kampmeier wanted to say something to my face, he should have called *me*, not **MacWEEK**. As stated in the review, I am in the Department of Computer Science at NCSU.

I agree that we need some face-to-face comparisons of rapid-prototyping and quick-development tools. If Dan would like to send me a copy of his interface product (FaceIt), I will consider writing a review of it.

But I do not guarantee a happy Kampmeier.

[I also feel we need a good comparisons of the current tools out there for this type of development need. CASE, computer aided software engineering, is a blessing as the learning curve goes up on new systems.-ed]

Volume 4 Number 11

Font Sorting via the Toolbox
David Dunham
Goleta, CA

In reference to Ray Cameron's Font Dialog Box article: Believe it or not, there's a toolbox routine which sorts font names (at least with 128K ROMs). Just have a fake (i.e. never displayed in a menu bar) menu, and use *AddResMenu (menu,'FONT')*. This does the right thing (or at least the official thing) with FONDs and NFNTs, and properly ignores fonts starting with "." or "%" (such as bold or italic screen bitmaps for PostScript fonts). If you want to use this information in a list just call *GetItem()*, *CountMItems(menu)* times.

Background Problems
David Linder
San Carlos

First of all, I want to apologize for the "Stone Age" way that

I am contacting you. Unfortunately, I have not purchased a modem yet for my Mac, so I am unable to reach you via a BBS – however, that will soon change...

Being a recent Mac convert, let me add to the accolades of others in congratulating you on a fine magazine. For those of us who are caught in the middle of the steep learning curve for Mac programming, MacTutor is a much needed resource. Keep up the good work – you can count on my subscription for as long as the magazine continues!

As one of my early learn–to–program–the–Mac projects, I tried working with Peter Korn's GraphLoad program from the March 1988 MacTutor. After typing it in, compiling it, and debugging it (using MPW C 2.0), it seems to work just fine when the GraphLoad window is the front window. However, when I activate a window from another application in Multifinder, GraphLoad freezes until I reactivate its window and bring it to the front. I thought GraphLoad would run in the background so that it would always show system load even when it was not the front window. What am I doing wrong? My SIZE -1 resource has the following bits set:

Accept Suspend Events
Can Background
Juggler Aware

Doesn't the Can Background bit mean that this program will run as a background program? Any help you can give me would be appreciated. *[Checking the source code as distributed on the source code disk #30, the program runs fine in MultiFinder. I'm looking at it as I type and it's blinking away. You must have an error in your code. Check it against source code disk 30. -Ed]*

Developer Responsibility
Edward J. Groth
Scottsdale, AZ

After seeing the DCM promotional ad on page 23 of the June, 1988, MacTutor and being quite disenchanted with the horrible implementation of the MS FORTRAN 2.2 package, I ordered and received DCM's MACTRAN PLUS.

I sent a letter to DCM about a month ago, and since I have not heard anything from them, I must assume that they have decided to ignore it. The note that came with their package that is referred to in my letter reads as follows:

> The file "tlbox.Rel" in the folder "Other files" is corrupted. You would have to reassemble the program "tlbox.asm" with a 68000 assembler. If you do not have one, please send the disk back to us, and we will include an uncorrupted, working copy.

I have sent you this letter for perhaps one or both of two reasons. I would like to see MacTutor supply useful and complete reviews of such things as languages. The usual magazines do not review languages but only applications. The user base needs such help. Hopefully reviews would include real comparisons of strengths and weaknesses of competing development systems.

The other reason is much more abstract. We really need to bring about some degree of maturity and responsibility within the software industry. This business of selling a blister-packed and copy protected piece of software without any responsibility to make sure that it works, at least somewhat, has got to go. There is also no recourse against extravagant and fraudulent claims made by many software developers and venders. I really have no good way to suggest for bringing this under control, but I am sure that exposure to the light of examination and comparison will certainly help. It seems to me that MacTutor is in the unique position of being able to supply such comparisons. I fully realize that this might not be viewed as the most astute business move, but I believe that being a real source of reliable information could only result in a more loyal readership.

Thanks for bearing with my frustration. If I can be of any help or provide any further information, please let me know.

[I, too, see no alternative suggestion except maybe the free market system. The amount of differing development systems and tools is just amazing, not to even mention the constant flood of upgrades. I personally do not trust a product until it has got a version 2 next to it. We will try, as space permits, to review development systems, but frankly, we do not get very many good reviews across our desks. -Ed.]

Disappearing File Icons and Other Problems
Steve Seaquist
Temple Hills, MD

When you drag a disk's icon into the trash can or select Erase Disk on it, Finder 5.3/System 3.2 and Finder 6.0/MultiFinder/System 4.2 sometimes forget how to redraw file icons on currently mounted HFS volumes. Here's what I've found out so far:

(1) Finder 5.3 doesn't redraw file icons at all and starts thinking that disks are too full to record folder changes.
(2) Finder 6.0 and MultiFinder temporarily draw generic file icons, then suddenly gray the desktop, pause briefly and redraw the correct icons. (Presumably they read the desktop file during the pause.)
(3) It's a consistent property of the ejected (or erased) disk.
(4) Every disk newly formatted by my 5.3, 6.0 or MultiFinder (800K or 400K) starts out with this property:
  (a) Erasing a normal disk causes that disk to exhibit the problem when its icon is dragged to the trash in the future.
  (b) Erasing a disk that already has the problem causes the icons to disappear as part of the erase process, and doesn't fix the disk.
(5) A visible characteristic of such disks is the fact that the Finder doesn't recognize the space taken up by the directory and desktop. (It says empty ones have "0K in disk".)
(6) It'll happen every time with the same disk until something fixes it.
(7) Once fixed, it'll never happen again unless you erase it.

Out of paranoia, I shutdown or restart when this happens. I haven't lost any data yet, but it's very annoying. I don't know how other versions of the Finder/System behave in this regard.

Fixing a disk with the disappearing file icons

In the process of testing whether Finder 5.5/System 4.1 had the same problem, I discovered that simply inserting the disk under 5.5 was sufficient to repair it and keep it from screwing up

5.3, 6.0 or MF. Although 6.0 and MF recover from the problem more gracefully than 5.3, they don't fix the disk the way 5.5 does. It would appear that someone removed the 5.5 fix in the development of 6.0 and MultiFinder.

ImageWriter drafts of LaserWriter'd docs

This is a user question, but it has developer implications:

I often have to produce business reports to my clients describing what changes I made (or recommend making) to their computer systems. I use just about every snazzy Word 3.01 feature, including footnoting, formulas, intermixed graphics (snazzy for those not used to Macintosh), outlining, side-by-side paragraphs, style sheets, table of contents generation, etc. My clients want to see professionalism in both the interim and final deliverables of the report. Because I don't own a LaserWriter, I have to go to Kinko's and run off copies there. That's where the problems start.

When Word detects that the Chooser'd printer is a LaserWriter, it changes the font widths. (I use Times-12, mostly.) Because word wrap and tabs are affected by the width of text that precedes the right margin or tab stop, all the line breaks and tables get screwed up. As a shortcut to reformatting the entire document, I change the margins from 1.25" (IW) to 0.98" (LW). This keeps the word wrap about the same, but I still have to reformat a lot of tab settings. To save time and money that would be wasted doing this at Kinko's, I keep a LW driver in my System folder, Chooser it even though I don't have one to print to, and reformat it on my system. Then I go to Kinko's and run off a copy.

Now what do I do? Do I leave the document formatted for the LW, so I don't have to reformat it again the next time I take it to Kinko's? If I do, I won't be able to get a draft copy on my IW. Do I Chooser the IW and reformat it again, so that I can get drafts on my system? If I do, I have to reformat it again for the next deliverable. It's absolutely infuriating. My dilemma gives rise to several questions:

(1) Why should the Chooser'd printer affect font width in Word? It doesn't in MacWrite. Word has to be doing it on purpose. But why? This is a question of concern to all developers of applications that print, including myself. Despite what it says at the beginning of Tech Note 72 about printing to LWs the same as IWs, we all know there are some reasons to print differently (notably page size). But is there a good reason to change font widths for LWs?

(2) More to the point, how does Word change the font width according to the Chooser'd printer? I didn't see any FONT or FOND resources in Word's resource fork, so it doesn't seem to be playing any "APPL file overrides System file" tricks. The only thing I can figure is that it could install its own QD bottlenecks (like the November 86 MacTutor article on using StdText and StdTxMeas to support tabs in text edit), and that the LW driver's QD-to-PostScript bottlenecks end up using Word's bottlenecks that way.

(3) Does anyone know a way to get Word to image the page the same way for both IWs and LWs? I'll take anything: a secret flag editable with ResEdit, an FEdit patch, a disabling of the trick in question 2, anything. The direction of the change depends on the answer to question 1. That is, if there's a valid reason to make text take up more space on the LW, then I'd greatly appreciate a way to make it take up more space on the IW as well. If not, I'd greatly appreciate a way to make its LW output as condensed as its IW output. *[I suggest you make the following assumption: MS Word 3.01 does not work on anything except a Laserwriter printer. -Ed.]*

ResEdit 1.0.1 bugs under MultiFinder

Deleting a string from a STR# resource causes ResEdit to crash. My workaround for this problem is to run ResEdit under Finder 5.3/System 3.2 whenever I want to delete a string from a STR#.

Sometimes ResEdit doesn't see all resource types (or all resources within a type) in the system file. My workaround for this problem is to restart the Mac and to rerun ResEdit right after the restart.

Volume 4 Number 12

Prototyper™ vs FaceIt™ vs Programmer's Extender™

George R. Cossey
Author of Prototyper™

First let me say that the goal of Prototyper™ is similar to the other two products, but the implementation is very different. The goal for all of us is to make designing and building a Macintosh program less painful.

Prototyper™ has a built-in visual editor and simulator for interactive graphical design of the user interface, while the other two products provide no direct visual interaction in the program design.

Prototyper™ generates source code that uses only Mac Toolbox and does not supply any private or additional libraries for linking to. The other two products supply libraries to link to and provide their own toolboxes to interface to the Mac interface.

Programmer's Extender™ and FaceIt™ both try to make the Mac interface less of a challenge by using custom designed libraries that combine the functionality of multiple toolbox calls along with some canned control code. My objection to that method, as a programmer, is that canned libraries work for only specific conditions and if my program requires something a little extra that is not in the canned library then I have to replace it with my own code that uses the Mac Toolbox anyway. Prototyper™ generates source code that uses only the Mac Toolbox, so any "tweaking" that I have to do to a routine is easily added.

I commend both Programmer's Extender™ and FaceIt™ for developing their own interface libraries to make programming on the Mac easier. Programmer's Extender™ is a fine product that shows the dedication and high level of expertise of the designers. FaceIt™ is a fairly recent newcomer and has potential to provide the same level of functionality that Programmer's Extender™ does.

In summary, I developed Prototyper with two main goals in mind. The first was to make designing a program's interface like painting a picture in MacDraw, with objects that could be resized and repositioned visually. The second was to provide source code that used only the standard Mac Toolbox and not any special

libraries. Both objectives were realized with Prototyper™, and I feel it is in a different class than Programmer's Extender™ and FaceIt™ which use special libraries of functions.

[I commend Mr. Cossey on his excellent product. I like designing my interface at a MacDraw-type level. Prototyper is definitely not perfect. There are many things I would like added or changed. But this shows how much I like what Prototyper tries to do. I want to improve what it does, not just throw it out. I can see in the future that this will be the avenue of interface developing. The interactive feel of seeing your interface is very gratifying. Mr. Cossey, please keep improving Prototyper. I feel you have a product with lots of potential. -Kirk Chase, Assistant Editor, MacTutor]

Word 3.01 Word Wrap
Steve Seaquist
Temple Hills, MD

After much more sleuthing, I've determined that font widths on the screen do not vary according to what printer is Chooser'd. The reason why word wrap and tab stops vary between ImageWriters and LaserWriters is Word's variation of how it measures an inch:

- Chooser an ImageWriter, then Show Ruler, then get out a real-world ruler (a tape measure, for example) and hold it up to the screen. The inches that Word shows are notably longer than real-world inches.
- Now start a print job using Tall Adjusted. Afterwards, the word wrap point will change. Then do a Page Preview. Afterwards the on-screen ruler's measurements will also change, and the new ruler's inches will be a lot closer to (a tiny bit shorter than) real-world inches. (The requirement of doing Page Preview to change the ruler is probably a bug. It should change automatically to reflect current word wrap and tab stops.)
- I believe Word is using THPrint()^^.prInfo.iHRes to determine the number of pixels per inch in the on-screen ruler will accurately reflect the number of inches when the document is printed. This is the old aspect ratio problem.
    - Dots on the screen are square 74dpi x 74dpi.
    - Dots on IW, Tall are rectangles 72dpi x 80dpi.
    - Dots on IW, tall adj. are square 72dpi x 72dpi.

To answer my previous questions:

(1) There is indeed a good reason to change word wrap and tab stops when changing between IWs and LWs. The difference in aspect ratio would yield a different number of characters that could fit comfortably on a line. Because only the center or end point(s) positions of lines are guaranteed, the LW driver's QD-to-PS bottlenecks would generate lines longer than anticipated (left, right, or centered justified) or too compressed (fully justified). So the passage at the beginning of Tech Note 72 (about printing to LWs the same way as you would to IWs) is misleading.

(2) Contrary to my earlier suspicions, Word wasn't playing any tricks to change screen font widths. In actuality, printer resolution changes printout font widths, and Word simply adjusts its screen display accordingly.

(3) The trick to getting IW rough drafts of Word documents that are intended to be LW'd is to use Tall Adjusted. I haven't seen any mismatches yet, and I don't anticipate seeing any, because Word seems to use the same 72 dpi ruler in either case. Too bad Microsoft's hotline people didn't know this. It would've save some people a lot of sleuthing.

I'm tempted to use Tall Adjusted and LW fonts in all Word documents, so that the decision of whether to LW the doc can be postponed. Unfortunately, Tall Adjusted seems to slightly improve, but not eliminate, the problem where Best mode printing jams together words that are in LW fonts. This makes Best mode Worst and eliminates the qualitative middle ground between Faster and LW (a middle ground that still exists if the doc is in an IW font). My only guess as to what causes that problem is the facts that FScaleDisable implies FractEnable (Tech Note 92, page 2) and Best mode printing is scaled down from the double-size font if available.

More DB and C
Michael D. Sammer
Bartlesville, OK

Please let me compliment you on the fine journal, MacTutor. I enjoy reading it thoroughly, and it is very helpful. I program strictly in C, therefore, I would appreciate it if we could see more articles on C. This would be extremely helpful to me. Specifically, some articles on database management, hashing tables, and details regarding definitions of records in a file would be very helpful. Techniques for manipulating data back and forth from the disk to a data structure, efficient ways to mark the end of a record in a database would be helpful. I am also interested in a scheme for searching the database in regard to a group of different index parameters, ie, a list of all programmer's who use a Macintosh, the C language, and are between the ages of 30 and 40. Efficient code for this type of a problem would be extremely educational.

Thank you very much for your assistance. Please continue the fine work which you and your staff have delivered.

[The content of MacTutor is determined primarily by what people contribute and lately the c people have not been contributing much. To get more c into MacTutor, we need more c contributions. -Ed]

MyTENew Suggestion
Moshe Foger
Tel Aviv, Israel

I would like to comment about calling ROM directly through pointers published in the last September issue (Mousehole Report, p. 18). In my opinion the code would not work for the following reason:

When Lightspeed calls the MyTENew, it adheres directly to Pascal stack conventions (leaving room for TEHandle and pushing pointers to the two rectangles) and jumps to MyTENew leaving the return address on the stack. However in MyTENew, the compiler automatically inserts "Link A6, #0" which creates a new activation stack needed to access the function's parameters. The link instruction saves the previous contents of A6 on the stack. The JSR instruction also leaves a return address on the

stack; thus when the ROM is activated, it treats the saved A6 and the original return address as the required rectangle pointers and puts the handle in the pointer to ViewRect...

To correct it, I would suggest the following in MyTENew:

```
TEHandle MyTENew(Dest, View)
Rect *Dest, *View;
{
asm(
    UNLK        A6      ;Balance for Link from compilr
    MOVE        (A7)+,A3 ;POP return addr. to safe reg.
    JSR         TENew   ;Jump to ROM
    JUMP        (A3)    ; Return
    )
}
```

Animated Cursors
David Stoops
Philadelphia, PA

I am an engineering undergraduate at Drexel University, a school considerably devoted to Apple's University Consortium. I am writing in response to the heated discussion of animated cursors in the March and May '88 issues. In a rare moment of inspiration, I realized that animating the watch is a chore ideally suited for implementation as a VBLTask. Referring back to the June '86 MacTutor, which contains an article on screen switching animation using the Vertical Retrace Manager, I wrote the enclosed snipets of Lightspeed Pascal code that seems to do the job.

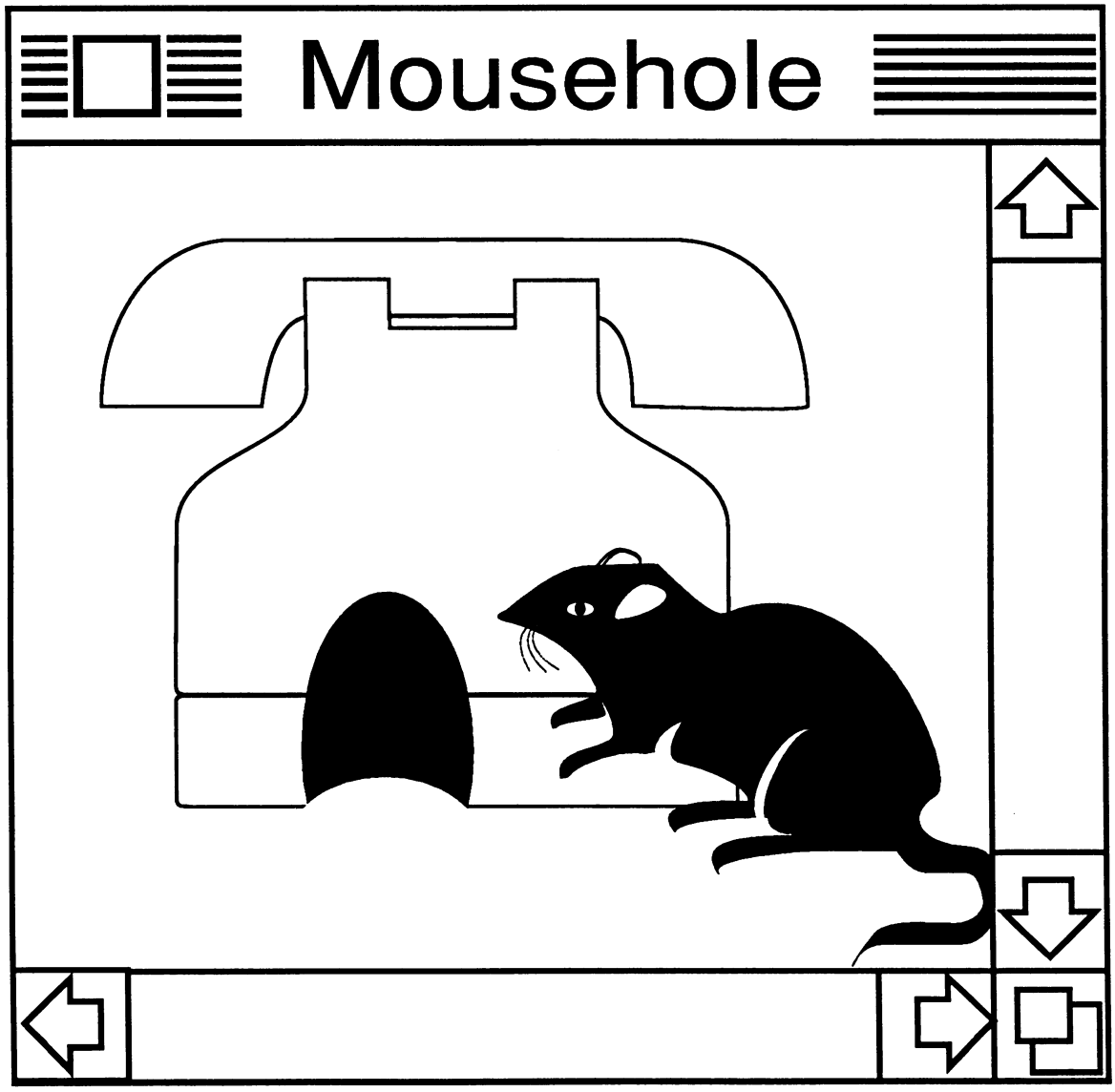
The routines make use of the 'acur' resource, which should be copied along with the necessary 'CURS' resources from the Finder into the project resource file. To use these routines, first call InitWatch to get handles to the cursors, then call StartWatch to display the animated watch while your program goes on about its business. A final call to StopWatch removes the task from the vertical retrace queue to halt animation. I find this scheme does a better job of animating the watch than that emploed by the Finder (notice how poorly the watch hands move during lengthy I/O operations under the Finder). Hope this proves useful.

```
unit WatchGlobals

interface

type
  acur = record
    whichWatch : longint;
    Watch : array[1..8] of CursHandle;
    end;
  acurPtr = ^acur;
  acurHandle = ^acurPtr;
var
  currentWatch : integer;
  WatchList : acurHandle;
  VBL : VBLTask;
implementation
end.

unit WatchUtilities;
interface
uses
  WatchGlobals;

procedure InitWatch;
procedure StartWatch;
procedure StopWatch;

implementation

procedure InitWatch; {Get the watch CursHandles}
  var
    watchcount : integer;
begin
  WatchList := acurHandle(GetResource('acur',0));
    {Will crash if 'acur' isn't available}
  with WatchList^^ do
    begin
      for watchcount := 1 to 8 do
        GetCursor(HiWord(longint(Watch[watchcount])));
      whichWatch := 0;
    end;
end;

procedure AnimateWatch; {Our VBL routine}
begin
  SetUpA5;
  with WatchList^^ do
    begin
      if whichWatch >= 8 then
        whichWatch := 1
      else
        whichWatch := whichWatch + 1;
      SetCursor(Watch[whichWatch]^^);
    end;
  VBL.vblCount :=10;
  RestoreA5;
end;

procedure StartWatch;
 {Install our task in the vertical retrace queue}
var
  err : OSErr;
begin
  with VBL do
    begin
      qType := Integer(vType);
      vblAddr := @AnimateWatch;
      vblCount := 10; {Update watch every 10 ticks}
      vblPhase := 0;
      err := VInstall(@VBL);
    end;
end;

procedure StopWatch;
{remove our task from the vertical retrace queue}
var
  err : OSErr;
begin
  err := VRemove(@VBL);
end;
end.
```

Carl Nelson & Associates Get a New Name

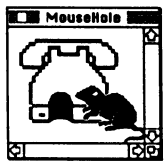
Our good friend, Carl Nelson, has changed the name of his company to Software Architects, Inc. Carl is head of the MacApp Developer's Association and is currently working on a number of important MacApp projects, including an article for MacTutor! Software Architects can be reached at (206) 252-6897 or by AppleLink D1846. They are located in Everett, WA.

A happy birthday to MacTutor, which officially completes four years of continuous publication with this issue. The first issue was published in December 1984, after MacWorld published their September cover story on how to clean your mouse! And the rest was history as they say. Thank you to all the faithful who have kept the programming dream alive, despite Apple's best efforts to frustrate us!



© The Definitive MacTutor, Vol. 4

Mousehole Report



Rusty Hodge
Contributing Editor
Mousehole BBS

Volume 4 Number 1

What: suitcase and MF
From: Venus (Jet Propulsion Laboratory)

This works just fine, but if (heaven forbid) you read your manual, in the current version of Suitcase, Open and Close DA/Font files are disabled. If I am doing something special that requires alternative DA/Font/Fkey/snd files, I boot with Command key down into Finder, reset the open ...Files, and then Option-Cmd-dblClk the MultiFinder Icon. It's a bit of a pain, but it works fine. I recommend stuffing as many resources into one file as possible and just working with it... I've got 36 DAs, 8 FKeys, 125 Fonts, and 15 Snds in one suitcase file named DAs and it works like a champ, using only one file buffer for all that. Faster booting too. (by some inconsequential fraction of a second). *[Note: we have found that Edit in the current release does not seem to work with more than 24 fonts in the system. -Ed]*

What: Flash command in HC
From: Greg Kearney (Howard Publications)

In HyperCard there is an undocumented command called FLASH; it is a XCMD that was added to HyperCard after Mr. Goodmans book went to press. The form is:

```
Flash <a number>
```

When called with out a number Flash will make the whole card "blink" several times. When FLASH is given a number, then it will blink the screen that number of times.

What: Control Panel stuff
From: Jim Reekes (CMS)

Opening a DA and getting only a beep usually means there is not enough memory available for the DA to live in.

Sam, what does 'rebuilding the Control Panel's desktop' mean?

From: Sam (Apple Computer Inc.)

The beep heard when opening the Control Panel is telling you that it is rebuilding its desktop. The control panel keeps a desktop file similar to the Finder's. It knows what icons to display and other fun stuff. Unfortunately, when you clear the PRam it also rebuilds the Desktop which takes a considerable amount of time if you have a large number of CDEVs.

What: Clearing PRam Sometimes Doesn't!
From: Larry Nedry (Mousehole Download Sysop)

As I said in a previous post, I can clear PRam with an older System & Finder but not with the latest. Have you tried it with the latest, Sam I have plenty of memory (4 Meg) so that isn't the problem. Anybody else had a problem with this ? I know of at least 2 others that have this problem also. What gives?

From: Power Hopeful (Maryland)

Larry, try giving the Finder more than the 160k that it claims. I also was having memory problems, but since I allocated 256k to the Finder things work fine.

I had an interesting and puzzling experience over the weekend. On receiving a new external hard drive, I was plagued by problems trying to set it up. I kept being told that there was no SCSI device connected in the programs I was using for that purpose. My elimination of the various things that might be causing this - such as changing the cables and even changing to a different Mac without an internal HD - had no effect on the situation. Since I have another external HD, I could pretty well eliminate the HD hardware as the cause since neither (both with terminators) showed up.

I knew it had to be a hardware problem, but I could not imagine what. Finally it occurred to me that I was using my new extended keyboard (which I don't like). Guess what: THAT was the problem! When I switched back to the old keyboard, everything functioned perfectly. *[?? I don't get this at all. Someone please check this out. -Ed]*

What: Radius Accelerator
From: John S. Lee (Argon Software Technologies)

Just bought some new toys for the Mac. The Radius Accelerator w/ 68881 co-processor: This seems to work fine with MOST software. I have not had a chance to run all of my software on it, but as expected there are NO problems with TML 2.x (I am using 2.1), and NO problems with LSP 1.11. RR 9.4 will NOT work with the Accelerator (or could it possibly be the new System and Finder). My DataFrane 20 (1 year old), needed some update software which I got today (Manager 3.21) and it works wonderfully with the Accelerator. I also have a Bering 20+20 removable and it works fine with the Radius. The drive will not boot up by itself, this is primarily due to the ROM versions inside the drive. Bering said they will ship them to me as soon as I write for them (strange, but true). Otherwise the drive works well with the Radius. Also got the Plus expanded to 2.5 for some leg room.

If I find any discrepancies with software and the Accelerator I will post it here. But so far it seems as though it is quite a solid piece of hardware.

What: Find File suggestion
From: Jim Reekes (CMS)

Use 'Find File' to locate the desired file. Type Cmd-M or choose 'Move To the Desktop' from the menu. Double-Click the icon to open it.

When your done with this file, at the desktop select the icon and choose 'Put Away' in the Finder's 'File' menu. Find File will also set the 'Standard File' dialog to the directory that the file is located in. This means that if you had used Find File in an application and located the file your looking for, then used the 'Open' file command from your application it would be inside

the folder that you had just found your file in. Make sense? Well, why ya think they called it 'Find File'?

I would like to see this option inside of the 'SFGetFile dialog. That would be a cool new feature. Think Technologies has an INIT file that does this.

Volume 4 Number 2

From: jimr (Jim Reekes)
Subject: add

The bogus FONT problem I've been complaining about with PageMucker also occurs in Illustrator. It is exactly the same problem where all the fonts, menus, and dialogs are totally unreadable.

I also had a similar problem today using MS Word 3.01 and the Background Print application in MultiFinder. I had a page of Palatino text, which I printed with "Fractional Widths" selected in the Print Dialog. As it began to print, the lower left corner of Word's window became garbage. This went away as soon as the print job was completed.

But the other strange thing is when I switch over to the Print Monitor to view the job in progress. It had the same problem that PageMucker and the rest have with FONTs, menus, and the dialog. Its all full of garbage.

From: jimr (Jim Reekes)
Subject: Re: FONT problem solved!

Okay, so I spent nearly an hour trying to locate the source of my problems with PageMaker, Illustrator, Word, Print Monitor, etc...

What I did was start with a fresh new copy of the system/finder. Then added my Laser fonts (which have been renamed to +Times, +Helvetica, etc.). That still worked fine. Then I added the fancy ImageWriter fonts that I've used for the past few years. They are Dali, Boston, Alexis, Dover, Florence, and probably another one or two. That's when I started getting into trouble with the above programs. Then I removed all of the fancy ImageWriter fonts, but I still had troubles.

So, started back with a fresh new system/finder. Added only my renamed Laser fonts and settled with that. Never use the ImageWriter lately anyhow. My conclusion is that the Font/DA Mover is scrambling the FONT resources. This is what got the above mentioned applications into trouble. Even after removing the offending fonts, the applications would remain screwed. So now I'm happy and using PageMAKER once again.

[In the November issue, we published a solution to font problems with Pagemaker. Check the FOND resource flag word with ResEdit and make sure it is set to $6000 for older Imagewriter fonts. Pagemaker is trying to read a non-existent width table in the FOND resource for these Imagewriter fonts and that is why the fonts are messed up. We've done this and have had no further problems with fonts in our system file. Pagemaker and Illustrator are the only two I know that use the FOND resources for calculating character widths. -Ed]

From: dirck (Dirck Blaskey)
Subject: Font bugs

A font bug: Starting with system 4.1 and continuing on into 4.2, the following occurred: An application has a font in it's resource fork (e.g. Times 9). If said font is not in the system file, a setfont will get a SCALED version of the font (e.g. Times 10) from the SYSTEM file. If the font IS in the system file, a setfont will get the font from the APPLICATION file. *[We recommend removing all fonts from application files. -Ed]*

From: davidw (David Whiteman)
Subject: Mac II problem with ADB

Has anyone heard of problems with the Apple Desktop Bus circuitry with the Macintosh II? Three Mac II's owned by friends of mine had to have motherboards replaced because the Mac would not respond to either keyboard or mouse input. This occurred after they switched from the standard keyboard to the extended keyboard, and subsequent attempts at trying different cables, keyboards or mice did not fix the problem. Has anyone else been having this type of trouble? *[I have the extended keyboard and the Mac II and no problem like this. -Ed]*

From: powerhopeful (Power Hopeful)
Subject: Re: Mac II problem with ADB

Yes, I've had problems like that. Mine seem too to interact with SCSI. Anytime I add anything new or change the order of SCSI devices, it takes me about a day to get things to function correctly.

From: rick (Rick Boarman)
Subject: Re: Mac II problem with ADB

I had to replace the mother board on my MacII for those exact reasons. I think what killed it is that I pulled the keyboard cable out with the power still on (I know, it was stupid to do that).

From: s.winders (Scott Winders)
Subject: PRam and System 4.2

Clearing PRam with System 4.2 is slightly different. The "Shift-Option-Command key down while opening the Control Panel" function remains the same. The difference is this: Only the PRam locations that can cause problems with the system are reset. These include: Startup Device, serial port settings, sound settings, and a few more. Things like the time, date, keyboard, mouse, and sound volume settings are not affected. This undocumented change has led to the confusion about this function not working.

From: jimr (Jim Reekes)
Subject: PRam and System 4.2

I just got hold of David Ramsey's new INIT file for Mac II users. It will protect us against the evil PRAM woes. I'm not exactly sure what it does yet, but David has expressed to me that he knows what is wrong with the PRAM and has solved it. This INIT will become part of the next system release, sometime in the first quarter of '88. I think I already saw this file on the download, I'll have to double check.

From: don (Don Melton)
Subject: PrintMonitor bug & MacDraw

I discovered a peculiar bug with MultiFinder's PrintMonitor a few weeks ago. It seems the good folks at Apple didn't set the default memory size large enough on PrintMonitor to make it compatible with their own MacDraw.

Here's the fix:

From the Finder, click on the PrintMonitor and choose Get

© The Definitive MacTutor, Vol. 4

Info from the file menu. Change the preferred memory size from 78K to 80K. Close the Get Info window.

That's it. Two stinking kilobytes of RAM! That drove me nuts for an afternoon. Has anyone else had this problem?

From: davidk (David Koslur)
Subject: PrintMonitor Size

Where did you get your copy of PrintMonitor from? Mine came with Apple's System Upgrade 5.0 and the size resource was already set to 80K. Someone may have played with yours before you got it.

From: don (Don Melton)
Subject: Re: PrintMonitor Size

I got my PrintMonitor from Developer Services like all the other certifieds around here ... and mine was set at 78K. Apparently they know I'm funny looking or something.

From: billr (Bill Rausch)
Subject: finder open command

One simple way to solve some of the folder confusion when using Multi-finder and Finder and other programs would be to have Apple modify the Finder Open command slightly. If no file is selected when Open is picked from the File menu, have Finder bring up the standard file dialog box with only applications listed. This way you could launch programs from MF without having to reorganize your desktop all the time. It would be useful from the regular Finder also although the Option-Open of the folders does the trick there.

From: geoffbryan (Geoff Bryan)
Subject: FKEY characteristics

Here is a question for the gurus (lsr?) — when an FKEY is invoked and its code is loaded, is that code locked? Assuming it isn't, isn't it risky to call anything that might move unlocked blocks around while the FKEY code is executing? Any opinions out there

From: lsr (Larry Rosenstein)
Subject: Re: FKEY characteristics

The system does lock FKEYs before calling them.

It should be OK for the FKEY to move memory. First, the very act of reading in the FKEY requires allocating memory. Second, FKEYs are run when the application calls GetNextEvent. GetNextEvent is listed among the routines that can move or purge memory. By extension, FKEYs are allowed to do heap allocations.

From: dmark (Dave Mark)
Subject: Restoring Device CLUTs...

This is a toughy (I think)! OK, first things first - I'm writing an application that directly accesses the video memory (instead of writing directly to a window). I then use SetEntries to load my own Colors into the current device's Color Table. My problem comes in restoring the original Color Table when my program is done. The likely candidates seem to be SaveEntries and RestoreEntries. Has anyone used these successfully? No matter what I try, my program crashes. Another method I tried was to save off the CSpecArray, and then restore it myself when I'm done. Unfortunately, when I look at the Color Table accessed via GetGDevice^^.gdPMap^^.pmTable^^.ctTable, there doesn't seem to be any valid RGBs (just a few -1s, followed by a whole bunch of 0s). Does this mean that a default of some kind is being used? I'm trying desparately to get something running in time for the MacExpo! Any and all help is greatly appreciated. I have the latest and greatest Volume V from APDA. You can reach me at NASA headquarter's during the day at (202) 453-8737. My name is Dave Mark.

From: lsr (Larry Rosenstein)
Subject: Re: Color Pix

I think CopyPix is superceeded by CopyBits, which can distinguish Bitmaps and PixMaps and does the right thing in either case. CopyPixMap has an entirely different function, I believe. I think this call just duplicates the PixMap structure, but doesn't copy the actual bits. (I don't have my Inside Mac volume 5 around, so I am going from memory.)

From: the_cloud (Ken McLeod)
Subject: CopyPix/CopyBits

That's what I think, too. I was under the impression that IMv5 was in a state of flux still, so I asked. CopyBits will do all the tricks I need it to, anyway...

From: dhill (David Hill)
Subject: resource fork size?

I am writing a program that has data files that store information in the resource fork and I need to be able to calculate the size of the resource fork so that I can display the file size from within the program. Does anyone know which call I use etc? Thanks-- Dave Hill (formerly the Corsair)

From: rsiegel (Richard Siegel)
Subject: Re: resource fork size?

You don't want to write data directly into the resource fork without going through the Resource Manager.

In any case, you do the following:

```
err = OpenRF(name, vRefNum, &path);
err = GetEOF(path, &count);
err = FSClose(path);
```

Now, "count" (a long) will contain the number of bytes in the resource fork. —Rich Siegel, THINK Technologies.

From: the_cloud (Ken McLeod)
Subject: Tear-Off Menus in LS

The tear-off menu code in December's MacTutor seems to have problems with the Lightspeed Pascal environment. (LS Pascal v1.11, System 4.2/6.0) The problem occurs when the tear-off window is initialized in the "main" part of the program: the windowDefProc field of the window (which is not visible at this point) is set to a pointer to the new window definition function. If this code is commented out, all works well within the LS environment (of course, the tear-off window will be a 'normal' documentProc type); otherwise, something gets confused badly. The patterns become garbage, and you can't quit (in fact, you're lucky to escape into Macsbug). Oddly enough, the code in the "MakeMenus" proc sets a pointer to the new menu definition routine in exactly the same way, and this works without a hitch. Also, if the code isn't run under the LS environment but merely compiled directly to application form, everything works as it should. I should also mention that I removed all toolbox initialization calls from the code (InitWindows, InitMenus, etc.), letting LS handle it. Can anyone (Rich?) shed some light on the

interaction of LS with new window definition routines (pertaining to Darryl's technique?)

From: lsr (Larry Rosenstein)

Subject: Re: Tear-Off Menus in LS

The technique used to install the defprocs (both the window and menu defprocs) trashes the heap. The code is directly modifying the master pointer. Memory blocks have back links which are used if the block needs to be relocated. It is possible that when you compile the program as an application it happens to work because the heap is never compacted; when running under LSP, you probably aren't so lucky.

The correct thing to do is allocate a 6 byte handle and place a JMP instruction in that handle. Then store the handle into the defproc field.

P.S. Although the article showed how to tear off a menu, it did not show the really tricky part: how do you make sure that the torn off menu always appear on top of all your windows? I would like to see a follow-up article to address this.

From: robh (Rob Humble)

Subject: Print Report Bug

Am I alone with this bug? I have two different stacks, each set up as a simple database, each with about 400 cards. When I attempt to do a Print Report, I often get blank pages in the report. Page 1 may have cards 1-25, page 2 will have 26-50, page 3 will be blank, and 4 will have cards 76-100. If I add or delete a card, the problem may go away, or may just shift to another page. An attempt to reprint the report without changing anything will skip the same page or pages. Also, the page skipped is not the same when the print is done on the IW or Laser.

I have talked to Apple directly, and sent them a copy of the stack. They confirmed the problem, forewarded it to Cupertino, and now I can't even get them to return my calls.

From: davidk (David Kosiur)

Subject: Mac-VAX SQL

Okay guys, run down to DEXPO and see Bob Denny demonstrate his latest feat of legerdemaine, SequeLink. SequeLink allows you to use 4D or HyperCard as a smart front-end to an SQL database on a VAX. It'll blow your mind! Makes "seamless integration" something that's here now!

Slowing down a bit — initially SequeLink supports 4D and HyperCard on the Mac, and Oracle on the VAX. A library for accessing the SequeLink driver on the Mac is part of the package, so you can write your own applications. Other SQL dbs will also be supported later. $395 for the Mac (including 4D runtime) and $1500-$10,000 for the VAX.

One last thing. SequeLink lets you run multiple processes to one or more SQL databases without a sweat. It's also MultiFinder-compatible.

Volume 4 Number 3

From: billr (Bill Rausch)

Subject: MacTutor article Bug

I just had a two page tech note printed in the January issue of MacTutor. It has an error. The address of the EventRecord must also be passed to the routines doing the event processing for the various window types. Sorry. Bill

From: jimr (Jim Reekes)

Subject: Finder problems

Is there a problem with the Finder or MultiFinder copying large amounts of files? Reason I ask, I couldn't copy my "Utils" folder that contained about 300 files (or 7.5MB) while running MultiFinder on a 4MB Mac II. It gives a message "Ran out of memory...." I tried to reboot into the Finder, but the same folder caused a BOMB=01 error with the said folder. Had to copy it in smaller chunks.

From: jck (Jack Kobzeff)

Subject: copying files in MF

It does seem that running with MultiFinder sure can limit the number of files you can copy. Even if there is no other program loaded, it looks like it can only use the memory allocated for Finder. I've run into this even with a fairly small number of files, you still get the 'out of memory' message. Whereas when running without MF, it copies several times the number of files without a problem.

From: rick (Rick Boarman)

Subject: Re: Finder problems

I had many problems using MultiFinder and large folders until I allocated 512k for it to use. After that, no more problems.

From: rusty (System Administrator)

Subject: Re: Finder problems

I've had a few 'ran out of memory' dialogs when trying to copy one hard disk to another under MultiFinder. Re-starting without MultiFinder made it work fine.

From: lsr (Larry Rosenstein)

Subject: Re: copying files in MF

There was a discussion of this on Usenet. The Finder uses the MF temp memory services for copying the files, but it has to use the Resource Manager to update the desktop file. This is the place where the Finder can run out of memory.

The Finder also uses a fixed size (200K, I recall) buffer for the copying, unless the copy involves removable media. Then it uses as much memory as it can grab to minimize disk swapping. However, there is a limit of about 40 file forks that it can copy in one pass, so if you copy a lot of files (or a lot of files with both a data and resource fork) then there may be extra disk swapping.

From: mark.chally (Mark Chally)

Subject: Multifinder file copies

According to the MultiFinder Programmer's Guide thing, the finder checks to see if MultiFinder global pool type memory (I don't remember the frigging terminology) is free. It then allocates as much as needed/or as much as it can. If you don't have any spare memory outside of the finder area, maybe this can be a big problem. Some programs can hog that other memory... It also said that if you don't have enough spare outside memory to be allocated by Multifinder, the Finder uses a small, spare buffer of its own. Don't know about errors though—I haven't had any problems—but I'm using four megs too...

From: jimr (Jim Reekes)

Subject: new desktop management

There is a 'Desktop Manager' that I've uploaded to the MHDL. It has been taken from the Apple Share system folder.

It is a MUCH faster method of keeping track of the desktop while in the Finder/MultiFinder environment. Anyone that is using a 80MB (or bigger) drive should put this file into their system folder and then reboot. After that, you can delete the desktop file. It is no longer used by the Finder/MultiFinder.

The new method of desktop management is to use B trees instead of the resource manager. This has already proven itself to me as a BIG improvement of the desktop's performance. Copying very large directories and lots of files only shows the "updating desktop file..." message briefly. Returning to the Finder is also quicker. Only drawback is that we can no longer unmount volumes by dragging the hard disk to the trash. Small price to pay. (Anyone can get this file from the Apple Share disks)

[Why didn't Apple distribute this file as part of the regular system 5.0 update if it is such an obvious improvement? -Ed]

From: dhill (David Hill)

Subject: multifinder Window Funnies

I found something interesting out about MultiFinder the other day. It will not put any of your windows in the front until your application calls GetNextEvent(). I found this out when I was calling SelectWindow() then ShowWindow() for my about window while the program was booting up. But when it would run under Multifinder, the about window would open behind the finder windows. The fix was to make a call to GetNextEvent before opening the about window.

From: jimr (Jim Reekes)

Subject: Re: multifinder funnies

I also found this out while trying to open a window during the boot process. I was using Dialog Manager calls. Until the program calls either GetNextEvent or WaitNextEvent, the program's layer doesn't come to the front. Ever seen a modal dialog in the background?

From: the_cloud (Ken McLeod)

Subject: 'acur' resource

Is Apple going to make the workings of the 'acur' (animated cursor) resource public? I don't see it mentioned in Inside Mac 5, but the latest ResEdit has a template for it...hmm... It would be great to be able to use animated cursors in other programs than the Finder! So...how 'bout it?

From: rdclark (Richard Clark)

Subject: Animated Cursors

If you examine the Finder's "acur" resource using ResEdit (some recent version), you'll see what the format is. But, to summarize:

<Number of Frames: Int>
<Frame counter: Int>
[repeat n times:<CURS ID:Int> <Filler:Int>]

Namely, a bunch of integers.

A typical entry would read:

```
Number of Frames: 2  (* This is a short one! *)
Frame counter   : 0  (* typical *)
CURS ID         : 128 (* First cursor *)
Filler          : 0
CURS ID         : 129 (* Second cursor *)
Filler          : 0
```

Writing a program to use this information is left as an exercise to the reader (although I may decide to do it myself some day...)

From: the_cloud (Ken McLeod)

Subject: Re: Animated Cursors

Thanks...actually I was hoping Apple had plans to implement this as a patch to SetCursor or something (that is, on a lower level than my having to write it!) so that if I set the cursor to one with an ID included in an 'acur' list, the animation would be automatic...

From: cderossi (Chris Derossi)

Subject: Re: Animated Cursors

Animated cursors are meant to perform two functions. First, they show the user that some lengthy process is underway, and he shouldn't panic. Second, the movement shows that the program hasn't crashed. But actually, the watch cursor itself handles the first requirement, so the animation is for the second reason.

This means that if you do the cursor animation as an interrupt process (like a VBL task, as some do), your program can crash and the cursor will still animate. So in order to use the animating cursor to show that the program is still alive, the program itself has to make the call to animate it.

And since that call is just SetCursor, addition System support isn't really needed. Granted, the System could help you figure out which cursor was next, but that isn't a very hard calculation.

Each of Apple's programs that animate the cursor (i.e. the Finder), do the work themselves. The 'acur' resource is just a handy mechanism for getting all of the data in a reasonable form.

From: the_cloud (Ken McLeod)

Subject: Re: Animated Cursors

You're right, of course... I tend to think more thoroughly about these things after my initial confused ramblings, but hey, I'm an impulsive poster. I do appreciate the answers and discussion generated about things that aren't necessarily documented elsewhere. It seems to me that the way to implement animated cursors must involve some sort of low-level installation, so that the repeated calls to SetCursor are asynchronous (something like a call to SetSound, for example..strike that..*StartSound*), and not just inserting a SetCursor(ID + 1) in every loop (assuming I have a loop to work with). Perhaps this will become ridiculously clear the moment I log off...

From: billr (Bill Rausch)

Subject: Re: Animated Cursors

Yes that is exactly what you do -> insert a SetCursor(nextID) into each and every place in the program where you need it. As it turns out, making it into a separate function SETCURSOR() isn't very tough. I use C, and just made the whole thing into a macro. Like Chris Derossi pointed out, setting it up as something automatic doesn't accomplish anything as far as the user is concerned. It is just as bad as using hardware blinking on terminals connected to minicomputers or mainframes with a message that such and such is happening. I've seen people wait hours on a dead program in situations like that.

From: royh (Roy Hashimoto)

Subject: Dumping PostScript

Does anyone know how to get the Laserwriter driver to

dump PostScript to a file instead of out the AppleTalk line? Is this an 'official' feature, or just a debugging back door? *[It is an official feature; just press cmd-K after clicking OK at the print dialog. This dumps the postscript and the LaserPrep header to a file named 'postscript0'. Cmd-F dumps without the LaserPrep header. Any cmd-K dump can later be sent via a postscript dumper utility and will print as expected. You can even edit the file as we demonstrated in a previous MacTutor, to get double column printing from data base programs that don't know how to format in columns easily. -Ed]*

From: mikesteiner (Mike Steiner)

Subject: FullWrite Pro eats memory

FWP really requires at least 2 megs (and more if you want to use it under MultiFinder). With a 1 meg Mac Plus, and TOPS active, there is not enough memory to even open a new document. With TOPS off, but with Suitcase active, there isn't enough memory to handle all functions. I had a "not enough memory" message when I tried to reshape a bezier curve and when I tried to resize a graphic. This was with the demo version they gave away at Expo.

From: markg (Mark Guidarelli)

Subject: Color resource support...

Ok I have a question...has anyone seen a program that will allow the designing of color cursors, icons, pattern or control items? I have looked around and none are to be found. One would think that Apple would have added that to ResEdit by now, but the most recent version of it doesn't support it. So unless one of you heroes out there has one or know of where I can get/buy(cheaply) one I'll have to write one.... It wouldn't be all that difficult but I would rather not re-invent the wheel (out of pure laziness). *[What you describe is not available, but COLORIZER from Palomar Software, available from the MacTutor mail order store, is a useful color utility that has some of this capability. -Ed]*

From: rguerra (Rich Guerra)

Subject: List Manager Question

I've been using LightSpeed Pascal to write an application that uses the List Manager. I create a list and draw it in a window. All functions (scroll bar tracking, highlighting, hit processing, etc.) are functional. A user double clicking on an item in this list causes another list to be created and drawn in the same place as the first, thus replacing it. This is where I run into a problem. This second list is functional in all ways save one: the scroll bar does not work. One can scroll the list by dragging the mouse, though with appropriate scroll bar tracking. I can even switch back to the original list which works perfectly. I know LClick is being called for clicks in the scroll bar of the second list. In fact, I create the second list exactly the same way as the first. This may be something obvious, but I'm at a loss. Any suggestions would be greatly appreciated.

I've tried the obvious things first:

The boundary rectangle for the second (and subsequent) list(s) includes the scroll bar (It is the same one that I used in the first list which works.) LClick does get called when I click on the non-functional scroll bar.

I turn drawing off on the first list [LDoDraw(FALSE,FirstList);] and also deactivate it [LActivate(FALSE,FirstList);] before swapping the second list into the display rectangle which is then activated and drawing set to true.

The lActive flag of the second (and subsequent) list(s) with a non-functional scroll bar is TRUE. The weird thing about this is that the scroll bar will track with the list when you drag on the items, still won't work when you click on it, though. Also, when the first list is swapped back in, the scroll bar works perfectly again! *[Sounds like you are not getting the proper control handle for the second list to your scrolling routine. -Ed]*

From: jaff (Mark Jaffe)

Subject: Re: Linking Desk Acc with MPW C

Sonny: I had similar problems with a cdev in MPW. What's happened is that you have some data that C puts into a DATA segment, and it's trying to use a (nonexistent) INITIALIZATION code. You probably have a STR or some other data. Everything used by Stand alone code MUST come from resources. If you are writing a DA, they must be referenced from the DRVR refnum. Suggest you study some other DA code examples (try Art Browser DA) Good Luck!

From: dirck (Dirck Blaskey)

Subject: LSC libs

To whom it may concern at Think... (Rich?) the realloc/relalloc functions in storage.c are USELESS!

From: dirck (Dirck Blaskey)

Subject: LS C compiler bug

LightSpeedC generates two (2) ILLEGAL INSTRUCTIONS in the following code fragment: t() { register long d; int i; d |= i; d &= i; }

From: robh (Rob Humble)

Subject: LSC Printing

I just go LSC at the show, and for a beginning C hacker, its great. But, the first project I have in mind requires access to some of the printing toolbox calls, specifically PrValidate, which is part of the print record. In looking through the docs I got (v2.15) I can't find any support for this. Am I wrong, is there a library that provides the support, or am I going to have to find another way to do what I want (determine the Chosen printer, a la Tech Note 72). *[Printing support is now in the ROMs, rather than a separate linkable print interface. See the Feb. issue of MacTutor for a Pascal example of printing a quickdraw picture. -Ed]*

From: the_cloud (Ken McLeod)

Subject: INIT Icons II

I just heard that the code for placing an INIT's icon on the screen in the lower-left corner at startup was placed into the public domain by its author, and people have been putting it to use, hence the proliferation of INITs that show icons. What is the "true story" here? Can anyone direct me to this code, if it exists?

From: lsr (Larry Rosenstein)

Subject: Re: INIT Icons II

The code to do this is called ShowINIT, and was written by Paul Mercer (now of Apple Developer Tech Support). He just released a new version at MacWorld Expo, which is compatible with a future INIT 31 resource that will perform the same function. You can contact Paul at: AppleLink: Mercer1 Delphi: PAULMERCER CSNET:

pmercer@Apple.COM US Mail: P.O. Box 160165
Cupertino, CA 95016

From: lsr (Larry Rosenstein)

Subject: Re: Tear-Off Menus in LS

I already wrote a TMenu object type that implements custom menus easily. Instead of worrying about defprocs, you define a subclass of TMenu and implement a couple of methods. The code for this is available on the 1st MacApp Developer Association disk. It would be easy to so a similar object type to implement custom windows.

From: thought.police (William Evans)

Subject: ApplScratch

Did I read somewhere that they were taking ApplScratch (the 12 bytes of low-address global space which can be used by an application for anything it wants) from us, or is that my imagination? It's probably my imagination, but if it isn't, I don't want to get bitten?

Any help in exorcising my hallucinations would be appreciated.

From: dorourke (David O'Rourke)

Subject: Re: ApplScratch

I'm not sure but from talking to a friend of mine at Apple you're not having delusions of grandure and the low memory globals you talk about are there.

They were originally documented in the 3 ring binder version of Inside Macintosh but were dropped in the "published" version. So far I haven't noticed any official reason not to use them, but you have to admit that's just too good to be true and since they aren't in the most recent version of Inside Mac I doubt Apple is going to support them in the future.

In fact I am guessing that Apple doesn't support them now, since Multi-Finder does switch {so I'm told} some of the Low Memory between applications. You might have a problem if you're running next to another application that uses those locations. It depends if MF switches and saves those for you, and if it does what happens if you try to access them while you're application is in the background. All sorts of nasty problems are associated with things like this so I wouldn't.

Now for some legend and some facts. Early versions of MicroSoft software reportedly used these locations to indicate if there had been a "key" disk inserted at an early time. Microsoft did this so that you only had to put the key disk in once, and then as long as you didn't turn the computer off it would remember that the copy you were running off of was alright. Now we're talking early Mac stuff here, when the 128K was the new kid on the block.

Now for the legend: Apple reportedly put those locations there at Microsoft's request, and in fact in early 3 ring binder versions of inside Mac there are some locations that have references to Microsoft.

Disclaimer: None of this information is official and I have no formal ties with Apple. Information presented here is to relate what I know and have heard over the years. I also have no association with Microsoft and I do not speak for them, and the information regarding them is also undocumented.

From: cderossi (Chris Derossi)

Subject: Re: ApplScratch

ApplScratch is (briefly) documented in Inside Mac, page I-85. This reference does not give its location, but it does say that ApplScratch will be preserved across ROM calls.

It also warns that ApplScratch must only be used by applications. INITs, cdevs, DRVRs, MDEFs, etc can't use this location. (MultiFinder must switch them across applications.)

From: jmunkki (Juri Munkki)

Helsinki University of Technology, Finland

Regions: The Real Story, a Technical Note

1. What We Thought We Knew

The May 1987 "ABC's of C" and October 1987 "C Workshop" in MacTutor provide descriptions of the region data structure. Obviously the region structure is reserved by apple and subject to change without notice (it's undocumented), but sometimes it's just too slow to do things the way apple intended us to do them. For this reason (and debugging purposes) it is useful to be aware of the structure. Unfortunately the "ABC's of C" article is not complete and the "C Workshop" gives slightly false information.

2. Conventions

In this note the structure I call the region is in fact the data that comes after the length word and size rectangle. The first 10 bytes are documented and are thus of no interest to us. While I'm on the subject of documentation, I might mention that the note in IM I-183 about the speed of OffSetRgn is misleading. None of the information in a region is stored in offset format and OffSetRgn is a relatively slow operation when the region is very complex.

3. What We Now Know

A region can be thought of as a collection of block descriptions, but each block is related to those above it. The block description starts with the smallest line coordinate included in the block and ends with a $7FFF. The words in between the start and end describe the difference between the current block and the one above it.

Let's examine a union of two rectangles: 10,10-100,100 and 80,80-200,200. The first line would then contain 10, 10, 100, $7FFF. This means that the first block starts at y position 10 and the pixels between x 10 and 100 are included in the block. The block extends until the block below it starts. The following block describes the changes to be made in the previous block. The second block would then be: 80, 100, 200, $7FFF. This means that those pixels that were between 100 and 200 should be inverted starting from y position 80.

The next block would then tell us to remove pixels between x positions 10 and 80 starting from y position 100. It should look like this: 100, 10, 80, $7FFF. The last block of our region would then cancel the remaining pixels from the block: 200, 80, 200, $7FFF. Since it is the last block, an extra $7FFF is appended. The complete region structure would look like this:

Data	Y	X Pixels set	Block
10 10 100 $7FFF->	10	10-100	Block 1
80 100 200 $7FFF->	80	10-200	Block 2
100 10 80 $7FFF->	100	80-200	Block 3
200 80 200 $7FFF->	200	None	Block 4
$7FFF			

In hexadecimal:
000A, 000A, 0064, 7FFF, 0050, 0064, 00C8, 7FFF,
0064, 000A, 0050, 7FFF, 00C8, 0050, 00C8, 7FFF, 7FFF.

Volume 4 Number 4

From: rajohnson (Robert Johnson)
Subject: Inside Mac & Palette Manager

Well, I finally got my long awaited 'final' copy of Inside Macintosh V, and I had very mixed feelings about the fruits of my waiting. On one hand,there was a lot of new information not in my APDA pre-release, and for the most part it seemed finalized and not contradictory (vs the aforementioned pre-release). But in no published documents have I found the trap numbers for the Palette Manager routines. For the method that is encouraged (almost at gunpoint) to access colors on the Mac II, the Palette Manager should receive extensive coverage. Even in the equates files included on floppy with the APDA pre-release, there is no mention of these trap numbers (I need trap numbers because I program mainly in assembly). Luckily, there is a place that I do have access to this information: MacsBug has a complete listing of all Macintosh trap numbers. Unfortunately the only information MacsBug gives to 'wh <trapName>' is the address of the routine patch in RAM.

Using the 'f' command, I was able to locate where in the trap table at $E00-$1E00 the particular routines were referenced, and each entry in this table is easily related to a trap number. In this way, I have compiled the list of unpublished trap numbers below.

Palette Manager Trap Calls

InitPalettes	AA90	AnimateEntry	AA99
NewPalette	AA91	AnimatePalette	AA9A
GetNewPalette	AA92	GetEntryColor	AA9B
DisposePalette	AA93	SetEntryColor	AA9C
ActivatePalette	AA94	GetEntryUsage	AA9D
SetPalette	AA95	SetEntryUsage	AA9E
GetPalette	AA96	CTab2Palette	AA9F
PmForeColor	AA97	Palette2CTab	AAA0
PmBackColor	AA98	CopyPalette	AAA1

```
PROCEDURE CopyPalette(srcPllt,destPllt:handle;
                      srcEntry,destEntry,count:integer);
```

is not documented anywhere that I have seen, but I deduced the preceding Pascal interface by disassembling the RAM patch. The procedure copies count entries from srcPllt starting at srcEntry to destPllt starting at destEntry. In the process of disassembling the patch, I found a slight bug in the code (I guess it's not that harmful in a non-documented trap). When srcEntry+count > pmEntries in srcPllt, it looks like the attempt to decrease count by srcEntry+count-pmEntries is botched because of a sign error (they subtracted pmEntries-srcEntry-count from count rather than adding it to count). If one is careful not to have srcEntry+count exceed pmEntries, all should be fine. This is such a useful routine; I think Apple should document it (and correct it).

Also, some other documented routines are missing their trap numbers in IM V, although they are listed in the APDA equates files on floppy:

CloseCPort	AA02	GetCTSeed	AA28
OpenCPicture	AA20	SetStdCProcs	AA4E

Of these, CloseCPort has been included in ClosePort (A87D) and OpenPicture (A8F3) will do the work of OpenCPicture when the current grafPort is color.

There is another trap I haven't seen anywhere: UpdatePixMap (AA38). Upon disassembly, it turns out to be just a RTS! Guess another system update is in order.

From: rick (Rick Boarman)
Subject: New ROMs

It seems that the new MacII ROMs cause several problems: TMON dies a horrible death. Version 2.8.1 is supposed to fix this. Macsbug 6.0 also dies. Dimmer 1.0B2 bombs. Anyone else have any problems yet?

From: adall (Alan Dall)
Subject: LaserWriter

Someone that I know has a Macintosh, a Laserwriter and many other computers of many types. They want to connect the Laserwriter up to a PC and then send their Mac output to the PC to be downloaded to the Laserwriter. They have created 2 files with the print option. One with command-k to generate the postscript header and one with command-f to generate the files themselves. When they transfer these files directly to the Laserwriter, all of their output comes out backwards. Does anyone know what they need to do to make their output come out right? *[You only need to use the cmd-K option, which creates both the file itself and the LaserWriter Prep header. That single file can then be downloaded to the printer and it should come out exactly like it would from the Macintosh. Something or someone is resetting something in the LaserWriter or else the prep file is not being executed properly so that the page orientation setup is being modified. -Ed]*

From: ewer (Bill Ewer)
Subject: Display PostScript

Does Adobe expect anybody to get all excited about Display PostScript? My first impression is that it's deadly slow. I estimate its max drawing speed is about 3K vectors a second. Does anybody know the exact drawing rate. How about QuickerDraw; how fast is it? The reason I ask is in advising a client on what display drivers they should include with their new color graphics app they are pushing for Display Postscript and I'm telling them its a waste of time and money. Any thoughts out there. *[I used to think like you until I heard Andy Hertzfeld say at the San Francisco Expo that Display Postscript is powerful and fast and that Apple should use it. That changed my mind real fast. If Andy says it is fast, then you can bet it's worth looking at. I saw it and it looked pretty good to me. I think Apple should make it an option. QuickerDraw is what Apple should have done in the first place if they had hired programmers who knew assembly language like Andy does, instead of Pascal coders. -Ed]*

From: powerhopeful (Power Hopeful)
Subject: Display Postscript

My hope is that it can be avoided. Regardless of its features, speed, or anything else, I think that its owner's licensing fees and behavior are ridiculous.

From: ericj (Eric Johansen)
Subject: Laserjet and Mac

To those of you who responded to my query regarding the Mac and the HP Laserjet, thank you. Here's the results. The goal was to find a solution to make the Laserjet, Mac compatible, and put it on the Appletalk network, so everyone could use it.

There is a driver called ProPrint, by Creighton Development. They're mentioned in this month's MacUser. I called them, the number's been disconnected, with no new number listed, i.e. they're history. One down. *[That was Chris Derossi's old place of employment! Chris is now at Apple Technical Support. -Ed]*

There is another driver by Softstyle called Printworks for the Mac, Laser Version. It kind of works, but I hate it. It was a pain to install. It uses 3 DA's to control such things as Font adjustment, Color Adjustment, and Spool adjustment. The Font adjustment DA is for matching screen fonts with what's available on the Laserjet. There's no postscript on the Laserjet, so it can't build different size fonts, other than what's supplied in ROM, Font Cartridges, or downloaded into RAM via a PC. The Color adjustment is automatically installed, even though on the "Laser" version, there is obviously no color printing. The Spooler that is included uses RAM. Uh-huh. It is not compatible with TOPS, both old and current versions, nor is it compatible with any other spoolers. In short it is a real pain in the rear end. The good thing I can say for them, is that their tech support people are pretty good, and more than willing to help with any problems. Two down.

Another option was the QMS board that makes the Laserjet postscript compatible. It's expensive, (around $2500), and it's not Appletalk compatible. Three down.

The Grappler LQ. One of you mentioned it here on the board. I'd heard about it before and they demo'ed it at MacWorld, but it's not shipping until March. However, after talking with the sales people and the tech support people at Orange Micro, this seems to be solve the compatiblity issue. The Grappler LQ enables the Laserjet to mimic the Apple Imagewriter LQ. You use the Imagewriter LQ driver that Apple is shipping, and Orange Micro's serial to parallel converter. The Laserjet "smooths" out the 240 dpi of the LQ and prints it at 300 dpi. Going by some sample copy I saw, this looks pretty good. Apparently this thing will make most popular parallel printers work with the Mac.

As for putting the Laserjet on the Appletalk network, I found the NetSerial by Shiva. This little sucker is expensive, (retails $399), but it works. It will put any serial device on the Appletalk network. *[About time someone invented that! -Ed]*

Since the Grappler LQ is a serial device, the combination of these two should work. God willing.

From: jsurreal (Roy M. Lovejoy)
Subject: List Manager Bug..

I ran into what I think is a related bug in the List Manager.. This was my problem: I had an application that created a window, then added a List to that window. When I read in a specific 'file' it would add the data to that list, when I closed that 'file', I did an LDispose, *BUT I KEPT THE WINDOW!* after three LNew-LDispose's on one window, the program hung... Delving deeper with a debugger, I found that it was hanging in my UpdateProcedure, during a DrawControls(MyWindow) {My Window also had buttons}.. Further investigation revealed that DrawControls was stuck in a <get-this> CIRCULAR LINKED LIST OF CONTROLS. It seems that either LNew or LDispose (probably the former) does not insert (/add) the List's scroll bar controls properly to the windows linked list of controls... My solution was simple.. I just deleted the data, not the ListHandle.

From: rguerra (Rich Guerra)
Subject: Terminal Emulation in LSP

I'm fooling around with Lightspeed Pascal trying to write a simple terminal program. I've got the basics working; now I want to add some other niceties such as screen buffers. Does anyone know how this is accomplished? To keep one huge bitmap corresponding to the number of screens around would seem ridiculously expensive in terms of memory. It seems from a previous message that using TexEdit is prohibitively slow. So how are buffers done? Perhaps incoming characters could be written to a chunk of handled memory and when a user wants to scroll back to see the previous screens, one could calculate offsets into the memory and copybits the characters into a bitmap that corresponds to the single screen normally displayed. Does this sound reasonable? Any thoughts, comments or suggestions would be greatly appreciated.

From: frank (Frank Henriquez)
Subject: Re: Terminal Emulation

I've written some generic low level serial I/O routines in assembly, and a terminal emulation program (both in Turbo Pascal and assembly) that makes use of these routines. My first version used DrawChar to display the text on the screen. DrawChar is fast, but it's a pain to do simple things like backspace.

TextEdit is slow, even with assembly language, but there is a trick that'll let TextEdit keep up with data coming in through the serial port: make the default serial I/O buffer larger (the default setting is something like 64 bytes). By making the buffer 1 or 2K big, TextEdit will not cause the serial drivers to loose data, and you can go to pretty high speeds; I've tried it at up to 9600 baud with no loss of data. I'm thinking of sending it all in to MacTutor, since accessing the serial ports has always been an unpleasant task and it seems to be one of those perennial Mac programming questions. *[Please do! -Ed]*

From: mark.chally (Mark Chally)
Subject: SetIText from a Desk Accessory

My main question is: Why does SetIText respond differently from within a desk accessory than it does from within an application? I wrote an application model of my "HexFlags DA" before doing the DA version. When I used SetIText to set the text of an edit Text field, it worked fine with the application. How-

ever, when I tried it from the DA, I found that the contents of the field were getting updated, but the CHANGE was not getting drawn. If I forced an update on the field, it would happen correctly. This was very frustrating as the performance of the Desk Accessory "looks quirky" as I found the "best way" to handle the situation was to do an "EraseRect" on the field's Rect.

I suppose I could do something more elegant, but maybe someone else can more accurately explain the problem and an effective workaround or fix? I can supply the source code in full to Apple Tech support for both versions of the program if it will help.

From: lsr (Larry Rosenstein)
Subject: Re: SetIText from a Desk Accessory

Here's the problem. SetIText does get passed the window. Therefore the trap has to scan through all the window looking for dialog windows. For each dialog window, it looks through all the items to see if the handle matches.

When it finds the right dialog & item it can update the screen. The text is always inserted into the handle, even if SetIText can't find the item to do the update.

The problem is that dialog windows are identified by a special value in the windowKind field. In a DA, this field contains the negative of the DA refnum. The solution is to save the windowKind and set it to dialogKind (=2) before calling SetIText. Then restore it afterwards.

From: atom (Mark Adams)
Subject: mpw LDEF's

Does anyone have any example code for writing a LDEF resource from MPW Pascal? I have converted code from Lightspeed pascal, and have gotten it to compile into a LDEF resource, but every call to the ldef by the list manager gives me an illegal instruction trap. I've looked around with Tmon, and from what I can tell, the pascal compiler is not setting up the code so that the first instruction in the code is the first instruction to execute. There seems to be garbage at the beginning of the code resource.

I did notice that if I took the variable declarations out of the LDEF procedure heading, (made it just Procedure MyListDef;), it executed the code correctly. Except of course then I had no variables being passed to the routine, so I couldn't do much.

Is there a compiler option or linker option that I need to set to get it to set up the entry code correctly with a code resource that need parameters? or can it not be done from pascal?

From: rdclark (Richard Clark)
Subject: MPW Pascal LDEFs

Look at Tech Note#110 "MPW: Writing Stand alone c Code in Pascal." BTW, you don't have to create a LDEF as a seperate resource... if you look in the List Manager chapter of IM4, you'll see that one of the fields documented there is a Handle to your LDEF procedure. You can make the LDEF a procedure in your program, then create a "dummy" handle (the size of a pointer) which points to your procedure. Then install this handle into the List's record. (confused?? Need the tech note?? Send me eMail with your SnailMail or FAX address, and I'll send a copy of the tech note and some sample code. Alas, the sample is in Lightspeed C, but it shows some of the tricks I'm describing.)

From: lsr (Larry Rosenstein)
Subject: Re: MPW Pascal LDEFs

Beware of creating "fake" handles. If you make a handles that's simply a pointer to a pointer to your procedure, you won't run under A/UX and risk being incompatible with future systems. The correct way is to create a 6-byte handle with NewHandle and put a JMP $xxxxx instruction in the handle, which jumps to your procedure.

Also, if you are doing a WDEF in this way, be careful to test it with MultiFinder. When MultiFinder calls a WDEF, it does not swap in the process associated with the window. Therefore a WDEF cannot easily access global variables.

From: atom (Mark Adams)
Subject: ldef handle

I used to do it that way (with a fake handle), but had lots of problems when I started allocating a lot of memory. And I never did feel right about it, since it isn't really the way to do it by Apple's guidelines.

From: the_cloud (Ken McLeod)
Subject: STRS resource

I have noticed that LightspeedC stores strings declared in a program (as opposed to strings gotten by GetString/GetIndString) in a resource of type 'STRS'. The format appears to be quite simple: a length byte followed by the string itself. However, changing one of the strings and adjusting the length byte accordingly (using ResEdit) instantly rendered the program unuseable. Is the size of the STRS resource taken into account somewhere else (like in a CODE segment, perhaps)?

From: dhill (David Hill)
Subject: Lightspeed C Multifinder bugs

Here is a list I have put together of problems I have been having with Lightspeed C under Multifinder.

1. On compile - gives an "illegal token" error in Mactypes.h, will not compile until I quit and re-run lightspeed
2. On Quit (or ShutDown) - gives a "Volume not found - PrintMgr.h" error, need to hit quit again for it to quit.
3. On compile - gives a system error $7FFF before the compile/lines dialog even appears, need to reboot.
4. On run - doesn't recheck the available memory, will not allow you to run even if you have plenty of memory currently available, need to quit lightspeed.

This is just to see if I am the only person having problems with lightspeed under Multifinder and hopefully to see some of these fixed in the next update. I am using Version 2.13 and have never had a problem running under finder.

From: dirck (Dirck Blaskey)
Subject: Re: Lightspeed C Multifinder bugs

I have no problems with lsc v2.13 under multifinder. Maybe your copy is corrupted. *[Current version is 2.15! -Ed]*

From: Steve Seaquist
Subject: New Virus Discovered

There is a new virus going around that affects MacDraw, Excel and others. It also leaves nasty messages on AppleTalk. This one is not peaceful, so watch out for it! It installs itself in the segment load table, after adding one to the number of code segments. It installs itself as the last code segment, which is one greater than the first unoccupied segment. It intercepts the start of program pointer in the segment loader, to point to itself so the segment loader launches it with a segment load. This comes per the Washington Apple Pie Bulletin Board, where the virus was first isolated. It is believed to have started in February of this year, but may date back as long as September of 1987. It also affects printing and set start-up options in the Finder. It is known to corrupt Excel files. It changes the type and creator of the note pad and scrapbook files such that the system folder sees them as plain document icons. It creates two invisible files called *scores* and *desktop* in the system folder. In the system file, watch out for inits of ID 6, 10 and 17. In the bogus system folder *desktop* file, there is an ATPL resource of ID 128, and a DATA resource of ID -4001 and INIT resource of ID 10. In the notepad file, the file type is changed to INIT and the creator to ZSYS, which is backwards. The virus knows about ResEdit and can re-install itself if you patch with ResEdit. Recommended solution is to restore all applications and system files that have been infected. The effect of the virus is to place bogus error messages in various appletalk dialogs, especially dialogs concerned with printing, thereby refusing to print over appletalk. The virus is detectable by *vaccine*, a commercial virus detector by CE Software, but not until after the infection. This is the same virus recently described in InfoWorld and MacWeek articles.

From: davidk (David Kosiur)
Subject: Fractional font widths

I just ran a cdev called Who's Who and it told me that fractional fonts was disabled on my Mac II. Question is — do any programs use fractional font widths? If so, which ones? Thanks. *[I believe Pagemaker uses fractional width tables. Ed]*

From: gking (Greg King)
Subject: SCSI comments

I just got another contract for a SCSI driver so I decided to update one that I wrote last year. After re-reading Jörg Langowski's article in the Jan/88 issue of MacTutor I decided to try out some of his suggestions.

One suggestion was to replace my looping TIB (transfer instruction block) with one that transfers all the data in one pass. It worked fine on my Mac Plus at home but failed miserably on my Mac II at work. The drive would always come up with a "Disk needs minor repairs" message and you could not read or write to it unless the operation was performed in extremely small chunks (<512 bytes?). Re-inserting the looping TIB allowed the driver to work on a Mac II. It is possible that the Mac II is so fast that a single instruction transfer overwhelms the drive and that a looping TIB gives the drive enough breathing room. Even with this deliberate degradation in performance a Seagate ST251N is best formatted with a 1:1 interleave on a Mac II. I have also found no difference in performance between the two TIB types on a Mac Plus.

Also, my July/87 copy of Tech note 96 contains no injunction against complex TIB's. What it does warn against is performing multiple SCSIRead's within a single SCSI command sequence (SCSIGet, SCSISelect, SCSICmd, SCSIRead/Write, and SCSIComplete).

PS: The documented word order for the longint result from KeyTrans in IM5 is backwards.

From: jimr (Jim Reekes)
Subject: Re: SCSI comments

Makes me smile. Greg, you've just realized *some* of the difficulties in developing SCSI drivers for the Mac. The Mac II will perform nearly 98% of the SCSI spec (about 1.4MB/sec). Trouble is, most drives will not come close to this speed. The 251N is closer to 800KB/sec. Blind operations are what typically hangs up the transfers. But I'm confused by some of your statements, I would have expected the Mac II to perform the operation and the Mac+ should have failed.(?) Also, I didn't understand what you meant by "<512 byte transfers". The smallest transfer would be a single block which is at least 512 bytes. But to agree with you on your assumption, it is possible to overwhelm some drives, such as the high capacity SCSI Rodime drives which cannot handle a blind transfer. It's not the TIB or that it is a single instruction that overwhelms drives, it is the amount of data being transferred and the speed of that transfer.

From: lnedry (Larry Nedry)
Subject: SCSI comments

Greg, You must have made an error somewhere in your code, because I have written a SCSI driver using a non-looping TIB that works on all of the Macs. I have been using it with very fast drives so that may be the difference.

From: dirck (Dirck Blaskey)
Subject: floating windows

The tear-off menus in the product I'm working on use floating windows. I don't know what the recommended method of doing this is, but I patched 3 traps to get it to work. There's also a couple of hooks into the event loop for fun things like cursor maintenance (automatically sets up the arrow cursor when over the menu) and other fun things like updating. The tear off menus use bit maps and rectangle lists in the resource instead of a separate mdef with drawing code for every different menu. In other words, it's a generic graphic tear-off menu mdef. If there continues to be a lot of interest and/or confusion on this subject, perhaps I can obtain permission to publish the code, maybe in a MacTutor article or on the mhdl. *[Please see the very good article in the April issue of MacTutor on tear-off menus and floating windows done in C. This article had the blessing of Apple Technical support, which reviewed it before publication so it is a sanctified approach. -Ed]*

From: markg (Mark Guldarelli)
Subject: New MAC II ROMs?

Hmmm, we got 3 new Mac IIs in at work this week and TMON does not function on them. We have version 2.8 and it

works fine on the other Mac IIs there....so I'm wondering what the difference is. Has Apple issued some new ROMs for the II finally? If so what did they fix/change? The NuBus problem perhaps? *[I've heard that there is a TMON 2.8.1 that fixes whatever the problem is. -Ed]*

From: brett (Brett Bilbrey)
Subject: New Mac II ROMS

Apple has seeded some developers with new ROMS and the newest Mac IIs rolling off the production line have new revs. An earlier post referenced TMON working or not working on different Mac IIs. Well first, the current rev of TMON is 2.8.1, and if you want to find out what version of ROMS you have in your MacII use 'sGetInfo' or 'Slot Investigator 1.2'. If you have the old ROMS you will see an error or old ROM report, if you have newer ROMs the version will be displayed. Note in the newer ROMS slot 0 now refers to the CPU! (i.e. 68020) Now before you run out screaming "I need new ROMS", WAIT. If you want to upgrade your MacII, wait till QuickerGraf and 24-bit QuickDraw become real. For now let the developers play with their alpha ROMS and work their bugs out. So far, nothing that has been changed is useful yet, no one uses 32-bits and the rest is cosmetic. When you will need 32-bits, a real ROM set will be available.

From: billr (Bill Rausch)
Subject: MacWrite 5.0

A guy in my building just got MacWrite 5.0 (and upgraded his 128!!! to a 512KE) and it doesn't perform as advertised. The spell checker won't work on a 512KE. If you start MacWrite, create a new document, and spell check it, it might work - sometimes did and sometimes didn't. If you do any file opening or saving you are out of luck. The error message says the clipboard is too large - this even if only a single word is selected for checking. Either poor testing or false advertising to claim the program works on a 512KE.

I did take his disk down the hall and it works great on MacPluses and SEs. Also, it failed on another 512KE with external 800K floppy and failed on a Hyperdrived 512KE. (The first 512KE had an HD-20 attached.)

Does anyone have any comments? Our dealer didn't.

I made sure to test it with a clean, Apple System Tools disk for Finder 5.3 and System 3.2 with no extraneous INITs, etc. and it didn't matter. The symptoms were always that everything except the spelling checker works fine (and that of course is why he wanted the latest version).

I'm trying to talk him into upgrading again to a MacPlus for various reasons and I think he'll do it, but this is still bad stuff from Apple.

From: billr (Bill Rausch)
Subject: Re: MacWrite 5.0

After posting my previous message I talked with Claris tech support (nice guy by the way). I described the problem, he admitted that they had one and had been looking for a solution unsuccessfully. We talked for a while and then he got real interested when I said that I could repeatedly duplicate my problems. Apparently I was the first complainer who could demonstrate a repeatable failure or success pattern. After taking a bunch of notes, he left to go try replicating my problems.

I got a call back the next day from a lady just going down a list of names. They found the problem and will send out new disks to 1) everyone? or 2) everyone who complains? - I'm not sure which.

She didn't say but my suspicion is that they forgot something simple like unloading segments not in use. They've implemented the spelling checker as a captive desk accessory and use the clipboard to pass the words back and forth. I really did like it on larger machines, by the way. It seemed pretty fast and very easy to use - just like MacWrite.

From: rick (Rick Boarman)
Subject: Help!

Help!! Does anyone know how to get small text edit boxes to work correctly with small boxes? I need to put Geneva 12 characters in a box that's 9 pixels high. No matter what I change the font Ascent and line Height to, it doesn't work. The text either jumps when I hit tab or draws wrong on an update event. What gives? *[Can't be done with text edit. Use DrawString first, then draw box on top. -Ed]*

Also, does someone know how to read PostScript fonts? We need to be able to read a font file and process the characters in the file in a random access method. Adobe hasn't been much help in this (They've been *unfriendly* about it actually).

From: dhill (David Hill)
Subject: printing problems

I am having trouble with my printing routines.

What I am doing is calling prOpenDoc, prOpenPage etc. and then the same code that updates my windows. I am also calling SetOrigin to center the graphics on the page. What happens is the first page prints out perfectly, but any subsequent pages (running the same code - with different data) are missing a big chunk - they have a big white rectangle in them. This happens for different print loops as well. I think it might be a problem with SetOrigin messing up the VisRgn or clip rect or something. *[Try not using SetOrigin, and instead position on the paper directly, as you would on the screen. Also, you might look at the Plotter program in the Feb issue of MacTutor which printed a hairline plot using code similar to what you are doing. -Ed]*

From: rsiegel (Richard Siegel)
Subject: Re: Lightspeed C MultiFinder.

You can increase (or decrease) the memory allocated to your project by putting a SIZE resource in the *project* file. Putting it in the resource file won't cut it, since the SIZE gets used before the project resource file gets opened.

From: alpha (Jean Thomas)
Subject: TRACKCONTROL/ASM

I have two questions on TrackControl:

Does anyone know the proper way to get to use action procs with TrackControl from assembly? What I'm doing now generates ID=1 error.

```
clr.w    -(SP)
move.l   VertScroll(a5),-(SP)
move.l   Point(a5),-(SP)
LEA DoThings,a0
move.l   a0,-(SP)
_TrackControl
```


© The Definitive MacTutor, Vol. 4

```
    move (SP)+,d6
DoThings:
    move #1,-(SP)
    _SysBeep
    move.l   (SP)+,a0
    jmp  (a0)
```

Is this the right way? (I think it's time for Pascal..) The above question is for manipulating the up and down arrows in a scroll bar. The action proc is DoThings.

I remember reading in IM that whenever the user clicks in either the up or down arrow, you should check the cursor to make sure it is still in the arrow. If it is not, scrolling should should stop. Is this done with TestControl? How would you incorporate it into the shell above. Am I dumb for trying to do this in assembly, or what? So much housekeeping... Thanks for any help.

From: frank (Frank Henriquez)
Subject: Re: TRACKCONTROL/ASM

Part of the problem in your code is this:

```
LEA  DoThings,a0
move.l    a0,-(SP)

it should be this:
pea  DoThings
```

This is from Dan Weston's "The Complete Book of Macintosh Assembly Language Programming, Volume I", page 189. If you're doing any sort of assembly language programming on the Mac, Dan's books (volumes 1 & 2) are almost as important as Inside Macintosh. APDA has them, so do some bookstores, like the B. Dalton chain, and the UCLA book store.

From: mark.murphy (Mark Murphy)
Subject: Re: using Imagewriter.II built-in fonts

To send ASCII directly to the Imagewriter in 4D, use Set Channel and Send Packet. There is an example to set the serial port to the Imagewriter on page 160 in the Command Reference manual. The example shows the call as Set Channel(0;10+3072+16384+8192). Then all you should need to do is call Send Packet(strexpr). Send Packet is documented on page 157 of the same manual. Hope this helps!

From: rusty (System Administrator)
Subject: Funny Drive Problems

Anyone having any sort of Hard Drive problems, bring them out in the open. One thing I've found is that many people have the same sort of problems, but no one talks much about them. Maybe we can save someone some grief.

One problem I've seen/heard happens fairly frequently (not frequent in general, but frequently in the scope of problems in general)-

The (MiniScribe?) 3.5" mechs in the newer Mac SE's are failing in the same way: Drive isn't recognized at all, but the drive blinks SOS, so most of its controller must be okay. Has anyone ever been able to recover data after a problem like this? What exactly is failing? The actual SCSI VLSI?

From: rusty (Rusty Hodge)
Subject: Accelerated Drives

According to Radius (in their Accelerator manual), certain hard drives will do unnatural things to themselves when booted if the accelerator is active. Anyone know which drives these are?

From: davidw (David Whiteman)
Subject: Re: Accelerated Drives

Rusty, among other drives, the Radius card is not compatible with the non-SCSI hard drives: Tecmar, MacBottom, Paradise, Apple HD-20.

From: sam (Sam)
Subject: Re: CMS SD-60 (ST-277)

If you are having trouble getting your Macintosh II to boot from a Seagate ST225N or 277N it might be due to the amount of time the Seagate requires to pull the heads back to the stop at track 0 just after power-up. The Finder will update the volume info at the last block of the disk - 1 when the volume is unmounted which in Seagate's case will leave the heads at the farthest point from track zero. The drive moves the heads slowly at power-up, because the heads are not parked, it doesn't know over which track they lie (it's also not a good idea to ram the head assy into the stop at full speed). Anyway, to the point. The Mac II will wait I think it's 20 or 25 seconds for any active drives to respond. Guess what.... the ST225N takes about 2 seconds longer than the Mac will wait — I'm not sure about the 227. Could this be the source of the non-booting SD-60 ?? Is the SD-60 an internal drive? If not, powering-up the drive a few seconds before booting the Mac will fix the problem.

From: jimr (Jim Reekes)
Subject: Re: CMS SD-60 (ST-277)

Huh? I don't understand your point. The SD60 is a 277N. The heads do recalibrate during every power up cycle. I guess it does take about 20-30 seconds for this to complete. The Mac II will continue to check the SCSI bus for a device to respond until the first one does. This means that no matter how long it takes the external drive to 'come ready', the Mac II will continue to attempt to boot from it.

This is NOT the case with internal drives. The above situation is also (darn it) not going to work if you insert a floppy to boot from before allowing the hard disk enough time to spin ready.

If you have the situation where you wish to boot from an internal hard disk and also have a bootable external hard disk, then you should set the internal's ID to 0. This will cause the StartupManager to allow a longer 'grace period' for the internal to come up to speed. Otherwise, simply remove the system folder from the external. (It is a bad idea having more than one system/finder anyway.)

If you turn on the external drive and then immediately insert a floppy to boot the Mac from, then you did not allow enough time for the drive to spin ready. This is true for all Macs and hard disks. If you turn on the Mac II with an internal hard disk, and quickly insert a bootable floppy, then you've got the same problem above.

From: d.goss (David Goss)
Subject: Time of Day

Does anyone know how to tweak the time-of-day clock, e.g. to slow it down or speed it up? Some computer clocks have a trimmer capacitor for that purpose, but I don't have any docs on

the Mac and haven't taken it apart to look.

From: powerhopeful (Power Hopeful)
Subject: PMMU

I got the 16 mHz PMMU from Motorola !

But I also have a SLIGHT problem: I can't seem to locate where it belongs on the Mac II motherboard. I can only see 2 empty sockets, both near the MPU, but neither is big enough to accommodate the PMMU, which is the same size as the 68020.

From: wayne (Wayne Correia)
Subject: PMMU/HMMU

The PMMU replaces the HMMU chip which is a big black square chip that sits just to the rear of the 68020 at location U67. Be sure to observe the polarity of the chip by noting the dot, notch, or key at pin 1. As you face the computer from above and the front, the pin 1 location is in the South East Corner. When removing the square package IC, lift each side slightly until it pops out. Remember not to break/lose/spindle/fold/mutilate this chip as you will need to put it back in its place if your Mac II main logic board ever needs to be serviced or replaced. To insert the new 68851 PMMU IC locate the gold colored 'Y' pointing towards one of the corners. This is pin #1. orient that corner so that it points toward the South East corner of the socket and lay it in the socket. If you are confused, look for the number '1' silkscreened in white on the logic board. Now take your thumb and press down evenly on the center of the IC.

From: powerhopeful (Power Hopeful)
Subject: PMMU

Thanks, I was able to install it.

BUT..

I'm having problems getting the Mac to use 32-bit addressing. I've messed around with it for awhile and have made a few conclusions. I know there was a bug reported concerning inability to recognize 32-bits on NuBus. Could this same bug be preventing acknowledgement of 32-bits of address in main memory?

The byte at $CB2 indicates which addressing mode is in use. Attempts to change that byte to $01 directly will cause a system freeze-up, which is different from a system error in that it cannot be handled by Macsbug and there's no info or recovery. The routine _SwapMMUMode is patched to a location in ROM, where there is right before your very eyes an instruction that writes an illegal value to the PMMU. In that routine the machine will tolerate the changing of the $CB2 byte. But when an attempt is made to write #0 to the PMMU reg, the machine again freezes.

[The Macintosh OS cannot use the PMMU because it cannot work in 32-bit mode due to 16-bit addressing restrictions in the memory manager and segment loader portions of the ROMS. Only A/UX can use the PMMU. -Ed]

From: powerhopeful (Power Hopeful)
Subject: Slots

Anyone know how I can directly access the Mac II slots? Is the fact that any of the (24- OR 32-bit) addresses listed in IM5 give me a bus error due to the bug?

From: cderossi (Chris Derossi)
Subject: Re: Slots

First of all, you need to know that the Mac II slots are not numbered in software as 0-5 or 1-6. Instead, they are slots 9-E. This means that the 24-bit address of the card in the first slot is 900000-9FFFFF. In 32-bit mode, its address space is F9000000-F9FFFFFF (16 times as much).

You can experiment with this easily. Assuming that you have an Apple video card in the first slot (slot 9), use TMON to change the value at the address 900020. This will affect the first pixel(s) on the screen directly. [The first 32 addresses are used as registers for the TFB chip.] You should not get a bus error.

From: philk (Philip Kutzenco)
Subject: Mac II screen shake

My MacII with Apple Color Monitor and Apple Color Display Card with extended memory has a problem. Most of the time, the display sort of gently shakes - maybe wavers is a better word. It doesn't seem to have anything to do with accessing the drive (a CMS PRO 102). It happens even when I'm just looking at the desktop. I don't think it is power related as I'm am on a power regulator. Can someone help? I really don't want to lug it to my dealer and leave it hoping he knows what to do. By the way, degaussing has no effect on the problem. *[My Mac II has the same problem, usually at start-up. -Ed]*

From: steve.sheets (Steve Sheets)
Subject: Mac // Video Card

Does anyone know how to change where the Mac //'s Screen Pixmap is pointing at. Basically I want to try some video page switching. If you had a Mac // video card (fully stuffed for 8 bit pixels) set for 1 bit pixel mode, you should have enough memory on the card for 8 seperate PixMaps. Someone awhile back told me that the hardware can do this, how do you do it in software?

From: cderossi (Chris Derossi)
Subject: Re: Mac // Video Card

Having not actually tried this, I know that I am missing some details, but the basic thing you will want to do is this:

Create a new GDevice with the NewGDevice call. Copy almost all of the fields from the MainDevice. (But you'll want to use [I think] a refNum of 0 and a mode of -1 to indicate that this is NOT a video card.)

Make your phoney GDevice correspond the the MainDevice in global coordinates, and then make it active. Make the MainDevice inactive. This will make QuickDraw draw to your device instead of the screen.

If you want, you can experiment with using different (maybe larger) rectangles for your GDevice. When you do this, you are going to confuse a lot of routines (which is why Monitors can't change logical location until reboot). I suspect that the best time to do this will be at INIT time, and you're going to have to mess with some low memory globals (like the screen size ones) if your device doesn't match the main device.

Finally, you'll need to create a PixMap that goes with your device, and allocate some memory for it. Then, when you want to display the contents of your device on the monitor, use CopyBits. You can even keep your PixMap in some fixed mode (like 8-bit/pixel), and CopyBits will covert to whatever PixMap applies for your monitor.

As I said, this is sketchy, kludgey, and unsupported. But if you get it working, I'd love to see it.

From: ericj (Eric Johansen)
Subject: Macs and Laserjets

To those of you who responded to my query regarding the Mac and the HP Laserjet, thank you. Here's the results. The goal was to find a solution to make the Laserjet, Mac compatible, and put it on the Appletalk network, so everyone could use it.

There is a driver called ProPrint, by Creighton Development. They're mentioned in this month's MacUser. I called them, the number's been disconnected, with no new number listed, i.e. they're history. One down.

There is another driver by Softstyle is called Printworks for the Mac, Laser Version. It kind of works, but I hate it. It was a bug to install. It uses 3 DA's to control such things as Font adjustment, Color Adjustment, and Spool adjustment. The Font adjustment DA is for matching screen fonts with what's available on the Laserjet. There's no postscript on the Laserjet, so it can't build different size fonts, other than what's supplied in ROM, Font Cartridges, or downloaded into RAM via a PC. The Color adjustment is automatically installed, even though on the "Laser" version, there is obviously no color printing. The Spooler that is included uses RAM. Uh-huh. It is not compatible with TOPS, both old and current versions, nor is it compatible with any other spoolers. In short it is a real pain in the ass. The good thing I can say for them, is that they're tech support people are pretty good, and more than willing to help with any problems. Two down.

Another option was the QMS board that makes the Laserjet postscript compatible. It's expensive, (around $2500), and it's not Appletalk compatible. Three down.

The Grappler LQ. One of you mentioned it here on the board. I'd heard about it before and they demo'ed it at MacWorld, but it's not shipping until March. However, after talking with the sales people and the tech support people at Orange Micro, this seems to solve the compatiblity issue. The Grappler LQ enables the Laserjet to mimic the Apple Imagewriter LQ. You use the Imagewriter LQ driver that Apple is shipping, and Orange Micro's serial to parallel converter. The Laserjet "smooths" out the 240 dpi of the LQ and prints it at 300 dpi. Going by some sample copy I saw, this looks pretty good. Apparently this thing will make most popular parallel printers work with the Mac.

As for putting the Laserjet on the Appletalk network, I found the NetSerial by Shiva. This little sucker is expensive, (retails $399), but it works. It will put any serial device on the Appletalk network. Since the Grappler LQ is a serial device, the combination of these two should work.

From: mark.chally (Mark Chally)
Subject: Mac to Hell(oops, PC) and back

The successful transfer of files between a PC and a Mac requires two good terminal programs, a null modem cable (RS-232 with lines 2 and 3 crossed), and a little patience. I occasionally transfer files between my MacPlus and a '386 MesS-DOS machine... I use the _ImageWriter_ cable directly plugged into the COM1 serial port of the PC compatible, ProComm on the PC (see me for that if you want), and MicroPhone on the Mac. It seems to work okay, at 19200 BPS... seems to me there's some way to get the Mac to do 57600 too... maybe it's Crosstalk that does that. Also, there has recently surfaced a Mac version of the ARC program that's so popular on the PC. In essence, ARC allows you to stuff a bunch of files together and compress them, in the same way that we're accustomed to using PackIt and StuffIt...I used ArcMac (again, see me if you want a copy) with relative success... the current version is BUGGY, and xArcMac.exe for the AT didn't even work... but using the real ARC program did, though it complained about a garbage header, which was probably all the "extra" Mac stuff thrown into it... anyway, files successfully went both ways.

From: jcom (Jim Comeau)
Subject: Mac to PC

I've used MacLink Plus with modem connections - slow-. I just got a 5.25 drive for my SE. You need to use Apple File Exchange to use the 5.25 drive. I was surprised when I found a very minimal set of translators were included. Oh yeah, now you can spend some money to buy each translator. But, if you have MacLink Plus you can use it to do local file transfers. The setup is to use AFE to do a binary transfer (no translation) between your Mac drive and the 5.25 drive. Then you use the rather complete set of translators that come with MacLink Plus to do the local translation. This setup works in either direction and it works FAST. If you have any large amounts of data I'd recommend it as the way to go. I haven't had any problems yet. The apple 5.25 drive will write to an IBM HD disk at 360 KB but the IBM machine won't know what to do with it. Make sure you use DD disks. Hope this helps.

From: billr (Bill Rausch)
Subject: Mac to PC

We use the TOPS network with 15 Macs and 3 PCs and everyone seems very happy. Excel reads and writes Lotus and Symphony files, and Word 3 reads and write PC Wordfiles right over the net without having to copy the files first. Of course, once you've read them in, you can Save As to put the resulting file (after your changes) either on the PC or your Mac and in any supported format. It is relatively fast, about like working off of local floppies, and relatively bullet-proof. We haven't had any kind of crash since TOPS 2.0 was installed. Also, we're trying out InBox from Symantec/Think and it is a great success. It is very easy to send someone a message and attach a file to it. For instance, you're on a Mac, run Excel, save the spreadsheet in 1-2-3 format, and then use InBox to send it to a PC user on the Appletalk network. He gets the message and saves the attached file on his disk. Then he uses 1-2-3 to read and manipulate it. Eventually he could reverse the process. (We're just now looking at PC Excel - it would make life even easier.) The nice thing about the E-mail approach is it doesn't require the simultaneous cooperation of two people using TOPS (one has to publish and one has to mount a particular disk or directory). Anyway, we're very happy with both products and recommend them to anyone looking for Mac to PC connections.

Volume 4 Numer 6

From: jvsossian (James Von Schmacht)
Subject: Re: Current Laserwriter version

Current version is 5.1 (for LaserWriter II's) and reportedly has some of the spool problems fixed. Driver is available on MHDL.

From: adall (Alan Dall)

Subject: Scores Virus Got Me!

I have aquired a virus on my system that seems to replicate itself on all of my applications. It creates files called Desktop and Score in the system folder. It causes MPW to not find the Worksheet file and may do other things. Has anyone come across this virus and does anyone know how to get rid of it?

From: Inedry (Larry Nedry)

Subject: Scores virus Effects

This virus creates an invisible file call Scores. This is an RDEV resource which executes at boot time similar to an INIT resource. Another file is created called Desktop which is an INIT resource. The Scores virus also attaches itself to the Scrapbook File and Note Pad File. If they don't exist they are created. These files need to be deleted. Also the virus attaches itself into the System file and every application that you run. The best way to get rid of this virus is to throw away the System Folder and replace it with a new one. The *Vaccine* INIT by Don Brown of CE software will notify you when this virus trys to add CODE resources to your applications. You can also use the *Aask* INIT to prevent the INITs from being executed while you are trying to rid your disks of this virus. I am writing an application this weekend to hopefully terminate this virus from your system. When it is completed I will post info about it here and upload it to MouseHole Download. *[This virus has been well reported in the press in MacWeek and Macintosh Today as the 'Scores' virus. It is dangerous so watch out. It was originally written to attack two internal programs at EDS in Texas and is said to be rampant in the Dallas area. Larry was quoted in the April 26th edition of MacWeek extensively about this virus and his Ferret program which detects it. -Ed]*

From: maxr (Max Rochlin)

Subject: Scores Virus Symptoms

The presence of the virus in the Macintosh memory does causes several symptoms, which have caused losses of data. These symptoms include difficulty running MacDraw, difficulty printing from any applications (especially MacDraw), difficulty using the "Set Startup" option, difficulty running Excel, corruption of Excel files, and frequent crashes when starting applications. This virus has existed since at last February, 1988, and may have been around as early as September, 1987.

It is possible to determine if this virus has infected your Macintosh with the following procedure:

1) Open the System Folder of the Macintosh and locate the "Note Pad File" and "Scrapbook File".

2) Examine the icons used on these files and check that they resemble the small Macintoshes seen on the "System" and Finder" icons. If they do not, and instead resemble the standard Macintosh document icon (an upright piece of paper with the upper right corner folded forward), you are probably infected.

3) To verify infection, execute ResEdit or some other utility which can see "invisible" files. Examine the System Folder.

4) If the System Folder contains two invisible files named "Desktop" and "Scores", you are definitely infected.

The virus transmits itself from Macintosh to Macintosh by invading a standard executable application file on a contaminated Macintosh. When this contaminated application is copied to a "sterile" Macintosh, the virus attacks the new system by making these changes to the System Folder:

• Three INIT resources are added to the "System" file. If the files "Note Pad File" and "Scrapbook File" do not exist in the System Folder, they are created. The type and creator fields of the "Note Pad File" are changed from "ZSYS" and "MACS" to "INIT" and "ZSYS", respectively, and an INIT resource is added to the file. The type and creator fields of the "Scrapbook File" are changed from "ZSYS" and "MACS" to "RDEV" and "ZSYS", respectively, and an INIT resource is added to the file.

• Two new invisible files are added to the system folder named "Desktop" and "Scores", each with an atpl, DATA and INIT resource.

Note that, unlike the MacMag virus, no "nVIR" resources are used anywhere. The modified files, "Note Pad" and "Scrapbook", still appear to function normally with the Note Pad and Scrapbook Desk Accessories, and any existing contents of the file's Data Fork are not disturbed.

As each application is attacked, the virus installs a new CODE resource into the application. The identification of this new resource is variable, depending upon the existing resources within the application. The virus looks for the first available CODE resource slot, then places the new resource one position above that. For example, HyperCard contains CODE resources 0 through 20, leaving an ID of 21 as the first available resource ID. The virus placed the new CODE resource in the application as CODE ID=22.

The second step of the infection of the application is the modification of the CODE ID=0 resource of the application. The virus modifies the eleventh word of this resource, which is the start of the application's jump table. Where the application would normally jump to the CODE ID=1 segment, the virus modifies this pointer to refer to the new CODE resource that has just been installed.

Note that the eleventh word has been changed from "0001" to "0016", which points to the new CODE ID=22 resource (hex 16 = decimal 22). Also note that during our examination of suspected applications, we found that at least one compiler - LightSpeed C, I think - normally places non-"0001" values in the eleventh word of the CODE ID=0 resource. To verify infection if the eleventh word is not "0001", check to see that the tenth word is NOT "4EED" and that the eleventh word points to another CODE resource. If both of these are true, then the application is infected.

The new CODE resource is a copy of the virus code, is of size 7026, and is executed when the infected application is invoked. When the virus completes execution, it returns to the invoked application, which appears to proceed normally. The first sixteen words of the virus are:

```
0000 0001 xxxx 3F3C 0001 A9F0 4EBA 002E 204D D0FC 0020
43FA FFEC 20D9 2091 204D     ...
```

The third word of the virus code is variable, and appears to

be based on the return address used when the execution of the virus is completed. The virus further modifies the code of the application in a manner which has not been fully deciphered.

If your Macintosh is infected, the contaminated system files and applications must be completely removed from the Macintosh, and new ORIGINAL copies should be installed. When removing the virus from the Macintosh system files, you cannot just go in with ResEdit and delete the offensive INIT resources - this virus is apparently intelligent enough to recognize this attempt, and modifies it's resource identification and memory location when probed by resource utilities. ResEdit "thinks" that the virus resources have been deleted, but they have been renamed and will return when the Macintosh is restarted. The system must be sterilized by:

1) Examine EVERY application (including any in the System Folder, and on EVERY diskette you may have) you have with ResEdit, and check if a new CODE resource has been added and if the CODE ID=0 resource has been modified to refer to the new CODE. This is the most tedious part of the process, and will probably take quite a bit of time.

2) Using ResEdit, open the infected System Folder and locate the "Desktop" file. Select the file and use the "Get Info" option on the "File" menu. When the file information window opens, turn off the "Invisible" bit, then close the window and save the file information. Do the same for the "Scores" file.

3) Locate a sterile system diskette (preferably one of the "System Tools" diskettes from Apple), LOCK IT, and boot from it.

4) Throw away the following files from the infected System Folder: "System", "Finder", "MultiFinder", "Desktop", "Scores", "Scrapbook File", and "Note Pad File". Once these files are in the Trash Can, EMPTY THE TRASH IMMEDIATELY! Note: this is the minimum required to remove the System portion of the virus - my personal preference is to delete the ENTIRE System Folder, not just the suspect files in it.

5) Locate all of the applications which you listed in Step 1. Throw them away, and empty the Trash Can.

6) Shut down the Macintosh, and turn the power off. Wait at least 30 seconds for memory to clear before rebooting again from the sterile diskette (this may not really be necessary, but better safe than sorry).

7) Reinstall the Macintosh operating system from the System Tools diskette to your Macintosh.

8) Locate your original copies of the deleted applications software. Before reinstalling the applications, examine each one with ResEdit to be sure that it is sterile. If there is no problem, reinstall the application.

A word of warning:

The *Vaccine* CDEV which is currently appearing on bulletin boards is only marginally useful in fighting this virus - if your system is already infected when you install Vaccine, you will not get any warning from Vaccine that the virus exists. If you have Vaccine installed on a sterile system, and this virus is introduced at a later time, Vaccine will only warn you of the virus attack, but will not prevent infection.

From: Inedry (Larry Nedry)

Subject: Scores virus Update

The previous description of the Scores virus is not entirely accurate. There are also a 'DATA' resource -4000 and a 'atpl' resource 128 that need to be removed. If you attempt to remove the virus and are unsuccessful it will modify itself and the previous instructions will no longer work.

From: wildman (Randy Saunders)

Subject: MacroMaker bug??

I just downloaded the 1.0b9 version of MacroMaker from MHDL. It seems pretty neat, much better than Tempo, but it did something wierd.

With MacroMaker in my System folder trying to print a MacDraw file causes an illegal instruction trap into MacsBug. Moving it out fixes the problem. I have: a) reloaded MacDraw 1.9.6 from my master disk; b) reloaded the Imagewriter driver (ver 2.6 4/17) from its disk; c) Throughly checked for viruses and found no unwanted visitors in my system. I am running System 4.2 and MultiFinder 1.0. I also have a number of other INITs, such as MFMenu, CPS Save Deletes, Front&Center, and of course Macsbug. Throwing out other guys does not fix the problem, although I did not try all the permutations. Is this a MacDraw 1.9.6 bug? (Other programs can print with MacroMaker installed) Is this a MacroMaker bug? If so, what could it possibly be doing to bust Imagewriter printing from MacDraw only?

From: jvsosslan (James Von Schmacht)

Subject: Re: MacroMaker bug??

I had the same problem printing in SuperPaint and VersaTerm. Nuked the puppy clean off my disk, and then reinstalled IW driver AND System.

From: mikesteiner (Mike Steiner)

Subject: STR Resource help

I'm almost to embarrassed to ask this question, because I'm sure the answer is so simple, but I can't figure it out. How do you change a 'STR ' resource from within a program? I can read the resource, place it in a editable field in a dialog, but I can't figure out how to write the change back to the resource file.

From: rdclark (Richard Clark)

Subject: Adding (and changing) resources

Mike, Take a look at Scott Knaster's Book "How to Write Macintosh Software". However, if you don't have a copy handy, I'll summarize what you need to do:

1) (This isn't in the book). Take a look at the STR# format, and notice that it contains an integer (the number of strings) followed by a set of Pascal-format strings (strings with a length byte in position 0). If you want to do anything that will affect the length of any of these strings, then you should take the STR# resource apart, make your changes and then put it back together.

2) Now that you have a handle to your modified STR# resource, we can do the update. The simplest way is to call

```
ChangedResource(theResource: ResourceHandle),
```

then make a call to ResError() which should return noErr (namely, 0). You may then call

```
WriteResource(theResource: Handle)
```

to make the change *now*, or the Mac will do it for you when you close the resource file (such as when exiting the application.)

3) On the other hand, if you have a completely new resource, you'll have to do something different. Take a look at the folllowing code-like example:

```
myHandle = NewHandle(100);  /* 100 bytes... */
/* Fill in the contents of myHandle here */

/* Now, we want to make this STR# resource #141, and
completely replace anything that used to be there.
So, we'll have to delete the old resource.
NOTE: if somebody goofed, we may have 2 or more copies of
STR# 141 already present, so get rid of all of them */

do {
    quit = false;
    if (scratch = 0)
        quit = true;
    else
        scratch = GetResource('STR#', 141);
        if (HomeResFile(scratch) = CurResFile()) {
/* If the resource is in our current file, then delete it
*/
            RmveResource(scratch);  /* Bye bye! */
        }
    else
        quit = true;   /* Exit if resource isn't in file
*/
    DisposHandle(scratch);  /* Don't clutter up memory */
    } while (done = false);
AddResource(myHandle, 'STR#', 141, ''); /* Add with no
name */
WriteResource(myHandle);       /* You now have a new
resource
```

From: mikesteiner (Mike Steiner)

Subject: Modifying STR resource

About 10 minutes after I posted that message, I found out what I had to do, and it worked. After getting a handle to the resource (TheHandle), I then did the following:

```
StrHdl := StringHandle (TheHandle);
StrHdl^^ := Name
(Name is the new string to go into the resource)
ResourceChanged (TheHandle)
```

Of course, I took care of resource purge and hLocks, etc. before and after doing the stuff, and I did have the proper semi-colons in place in the listing.

I need a bit more help. The method I stated in above works only if the replacement string is not longer than the original string. What do I need to do to be able to have a replacement string that is longer than the one it replaces in the resource?

I found my problem. I needed the following call:

```
SetHandleSize (theHandle, sizeof (str255));
```

and that solved everything.

From: the_cloud (Ken McLeod)

Subject: Yet Another Resource Question

I wrote a little CDEV that, among other things, wants to display the current System and Finder version numbers. As best I know how, you get the Finder version by doing a GetResource (or Get1Resource) on the MACS ID=0 string, taking the handle it returns and coercing it to be a handle to a string. Now the problem: it works just great everywhere EXCEPT when the Finder is open (which happens if the Control Panel is pulled down while in the Finder, or when running under MultiFinder). In that instance, GetResource refuses to give me any kind of access to the Finder ("File open with write permission" error). How can I get the Finder version number???? Incidentally, I get the System version number by calling SysEnvirons, so I don't have any open resource file problems there... Someone else has written CDEV called "Who's Who" that basically displays all the info from SysEnvirons. I tried it while in the Finder, and it gives me "Finder Version: Unknown," so apparently he didn't figure it out either. Anyone see what I'm missing?

From: rustyt (Rusty Tucker)

Subject: Re: Yet Another Question

I found one way to get at Finder resources, not quite painless though. This is a C frag but should be easy to convert to Pascal:

```
OpenFinder() {
int err, sysVRef;
HParamBlockRec  myParam;
Str255 myName;
    err = GetVRefNum(SysMap, &sysVRef);
    MemCopy( myName, "\pFinder", 7 );
    myParam.fileParam.ioCompletion = 0L;
    myParam.fileParam.ioNamePtr = myName;
    myParam.fileParam.ioVRefNum = sysVRef;
    myParam.ioParam.ioPermssn = fsRdPerm;
    myParam.ioParam.ioMisc = 0L;
    err = PBOpenRF( &myParam, 0 );
    FileErrorDialog( err );
    if ( !err )         {
        ShowInt( "\pOpened the Finder ", err );
        err = PBClose( &myParam, 0 );
        FileErrorDialog( err );
        }
}
```

The Mac will allow you to open the finder at this level with read-only permission. Natch you'll have to either copy the Finder RSRC fork into your own file then open it with the Resource manager, or parse the fork yourself and find the resource you want and make your own handle.

There is another way that should work, but I can't find enough info on MF to verify it. Switcher used to set up a Global pointer to the APPL zones it set up at $282. The first 8 words in the deref'd table had Ptrs to the "base" of those APP's. What in the heck does that mean? I don't know. Is it a pointer to the Heap Zone? If it is and MF maintains it you could just loop through that table and make SetZone(THz) calls followed by GetResource? Now that would be a lot easier than copying the darn Finder then doing a GetResource. So where are the other App's when your under MF? Inquiring minds want to know.

From: rickhyman (Richard Hyman)

Subject: CDEVs

Speaking of CDEVs (someone was), I created a small CDEV that included a EditText item in its DITL. This really screwed up the control panel; I took the edit item out and everything ran fine. Either the old RMaker is messing things up again or the control panel cannot handle all types of items possible in an appended DITL. I don't know the source of the problem. Have other people found this to be the case? If so, is it documented anywhere??

From: wbigelow (Wendell Bigelow)

Subject: 'BNDL' resources and the missing Application Icon

I've created an application and added my own Application icon and icon for the files it will create. I do not see the fancy icon in the Finder when I view by icon, just the default application icon. I've read several places about removing the "Desktop" file and otherwise forcing FINDER to rebuild the desktop and go out amongst ye Applications and (re)built of BNDL resources. This worked for a previous version of the application (and the files it had created) that uses the same icon image but has a different resource type (4 character doo-dah). I've deleted that particular BNDL resource and even managed to set the "bundle" bit in the GET INFO dialog in ResEdit for my application and forced desktop rebuild....ALAS...to no avail...

From: the_cloud (Ken McLeod)

Subject: Finder BNDL checklist

Did you: Make sure your application ICN# (and any ICN#s for documents created by your application) have corresponding FREF's? Make sure there's a corresponding resource with ID=0 for whatever your application's creator type is? Make sure the BNDL's owner is the same as this creator type? Rebuild the desktop by holding down the Option and Command keys when quitting to the Finder? *[The easiest way to get a new icon to appear is to use a new creator tag. Then the icon will be installed into the desktop file as soon as you use ResEdit to set the bundle bit. Otherwise, you have to get the desktop file to "forget" the previous icon for that creator tag, which is a pain to do. -Ed]*

From: allanc (Allan Clarke)

Subject: palette manager blues

I have an application which uses the palette manager. I keep a clut around and simply peel off copies (which are palettes) and attach them to the new graphics window(s). Unfortunately, my call to the palette manager does not seem to work on one out of about twenty-five Mac II systems (4.2). The call is...

```
myPH= NewPalette(256,myCLUTmTolerant,0);
if(!myPH) badnews();
```

I had my beta site do some testing for me. Here is the strange part. He boots off a generic system and runs a copy of the software off his hard disk and it works. He quits. He can now relaunch my application any time and it does not have any clut error.

When he restarts, though, back to square one. My application keeps ducking out with the Nil handle error.

What is the difference between booting off a floppy and moving to the hard disk versus booting off the hard disk? I have already had him pull all the INITs out of the way. MACDTS told me there were no error codes (QDError) usefully returned to give me a hint as to why the Palette Manager could not perform the call. *[You might check with last month's issue on Palette Animation to see if something is not being locked down properly. -Ed]*

From: lsr (Larry Rosenstein)

Subject: Re: palette manager blues Comment: to #49 by allanc

One thing that is different about the two situations you mention is that system patches are installed at boot time. It could be that the hard disk does not have the correct system on it, while the floppy does.

From: matt (Matthew Snyder)

Subject: system heap

The multitasker that I'm working on is written using a standard development kit, namely Mac C. Granted much of it is in assembly, I would still like to preserve the simplicity of using a standard compile-assemble-link-rmaker approach.

As it exists now, it is an OPTIONAL multitasker. You install it by double-clicking on its icon. It is loaded like an application into the application heap. It then does some extremely unmentionable things to move the barrier between the heaps and thereby sort of pull the covers up. It becomes part of the system heap and at the same time expands it.

Note: This is being used in an experimental and research-oriented environment, and there is really no intention of ever marketing it. As a result, I am not overly concerned with maintaining compatibility with future machines or future system software.

However, I am not happy with the complexity and the underhandedness of the whole thing.

The multitasker was originally ported from a different machine, where it also ran in the background, but where it used a native stay-resident facility. I was convinced that there must be such a native facility for the Mac.

I found that using resedit to mark the code resource as system- heap-destined would indeed cause the stuff to be loaded into the system heap, and I was trying to find out if I could turn this discovery into a stay-resident algorithm.

In response to your responses, I have another question. You say that the system heap can be expanded if you do it right, but that it can't be expanded while an application is running. What is happening while the loader is placing the next application into memory? If I specify that I want my code resources to be placed in the system heap when I get loaded, will the heap expand to accomadate me or not???

From: rdclark (Richard Clark)

Subject: Re: system heap

If you are using System 4.0 or later, you can resize the system heap at boot time using an INIT. The system recognizes when an INIT requests a block in the system heap and expands the heap if needed. (However, this isn't a nice thing to do as the Application heap shrinks. But, you already know that, right?)

And, I just remembered, there's another thing you can do! Instead of placing your code in the System Heap (requiring System 4.0++ and an INIT, you can place your code at the TOP of memory, up where only globals, the stack, and the ROM jump tables dare to tread. Look at "How To Write Macintosh Software" for an example of this. (Some RAM disks use this area, which is guaranteed to stick around between launches and which can be modified by an application. But, once you allocate the memory, it's unlikely you can ever deallocate it since somebody else may request a block below you and you don't dare move them. I doubt that this will create a problem for you.) This program wouldn't happen to be MultiMac 2.0, would it? <grin>

From: alpha (Jean Thomas)

Subject: XCMDS

Question: If you create a window with an xcmd/xfncs in Hypercard, how do you handle events in your window without

calling GetNextEvent? Whenever I do, events in my window are handled correctly but HyperCard is frozen until xcmd. If I don't call GetNextEvent, HyperCard is still active but I can't handle my window events at all. Any ideas? It makes sense, ie freezing Hypercard if you call GetNextEvent yourself, but how else could you tell when you are in your window? I suppose FrontWindow might do it alone, but I'll see. Any help appreciated. Has anyone else on this board written any Hypercard xcmds?

From: rickhyman (Richard Hyman)
Subject: Re: XCMDS

HyperCard was designed using its own event manager and memory manager; I'd like to know why because that really limits the kind of XCMDs you can build. You can handle events yourself by setting up a dialog for all of your user interface requirements. This points to the fact that XCMDs must be modal; i.e. anything your XCMD does must be completed, memory completely cleaned up, etc. by the time you complete your XCMD. Yes, you can use FrontWindow to find where you are, but there are situations where you can't be sure the user hasn't brought up the tool window, pattern window or message window. GetNextEvent doesn't work if you are using HyperCard buttons and text fields because HC does its own event managing. You can get away with creating dialog boxes and the dialog manager calls because that should be totally self contained in a single XCMD. It seems to be a good idea to flush events before handing control back to HC. If you are using HC for all of your user interface requirements, you may be able to break your code up into a number of smaller XCMDs. After spending 4 months on a stack and *then* running into these kinds of problems, I'm pretty frustrated with HyperCard. Not only does it need to conform to Mac interfaces from a user perspective, it needs to conform to interface and ToolBox standards as well. Unless it is cleaned up quite a bit, it will never be what Atkinson intended it to be.

From: alpha (Jean Thomas)
Subject: Re: XCMDS

I agree with you about the limitations of XCmds and external programming in Hypercard. Unfortunately, if you create a window with scroll bars etc, there is no way, at least I can see, to update your window without calling GetNextEvent in your code. SelectWindow is not enough because I don't know if HyperCard even knows when you have clicked on someone else's window. Perhaps you can send a series of idle messages to Hypercard. I don't know. Thanks for your help. I don't really think HyperCard was written for the traditional programmer.

From: rickhyman (Richard Hyman)
Subject: Re: XCMDS

The only way I found of updating various portions of my card, or specific windows was to use a series of transparent buttons on top of the appropriate areas. I experimented with mouseUp and mouseDown handlers to call the appropriate XCMD. This worked reasonably well for a list for which I used List Manager. If you are using multiple overlapping windows on a single card, then it would be difficult. If you have a single window with scroll bars, et al, then the use of several transparent buttons might work. But then, ... the several XCMDs would all, by necessity point to the same data structure for your window. So, you'd like to keep this structure in memory between XCMD calls, which HC could clobber any time ... and now we seem to be going in circles. Yes, HyperCard needs some work yet to be easily extended by programmers. *[Bill Atkinson demonstrated the next version of Hypercard at the Developer conference and indicated that a lot of internal bug fixes and changes have resulted in Hypercard both being much faster as well as the interface with dialogs being much cleaner. Some of these problems with XCMD's will go away with the next release, which should be out shortly. -Ed]*

From: lsr (Larry Rosenstein)
Subject: Re: Setting Startup from an app

Probably the safest thing to do is to write an application that launches another application based on some string resource. Then you should do Set Startup and choose the Finder (instead of MultiFinder) and choose to launch this application at start up.

By default, the string resource can specify MultiFinder, and then your machine will start up with MultiFinder. Later on, however, an application can change the string resource and have it start up with itself. You can make your startup program smart enough that it will default to MultiFinder if there is nothing else to launch. You might also make it do a sublaunch of the non-MultiFinder application (see Tech Note #126), and then launch MultiFinder when that application quits (or give the user a choice of another application).

From: lsr (Larry Rosenstein)
Subject: Tear Off Menus

I just read the Tear Off Menu article in the April MacTutor. It looked very complete and useful. I have only 1 minor complaint. On p. 32 the authors talk about the ToolScratch low memory variable, and the conflict between the Menu Manager and Lightspeed C. Inside Macintosh says that ToolScratch is not guaranteed to be preserved across calls to the Toolbox; it is designed to be scratch space for the Toolbox use. So the conflict is not the ROMs fault, it is LSC's.

From: wbigelow (Wendell Bigelow)
Subject: MacPaint/MacDraw 'PICT' files

HELP! I'm in the middle of implementing a nifty user interface. Unfortunately, I needed to create graphical objects I'll be displaying, in MacDraw and then save them as 'PICT' files. My hope was to somehow generate MPW compatible 'PICT' resource statements and manipulate these graphical objects as resources to be displayed in my nifty window design. I've got everything ready but I CAN'T FIGURE OUT HOW TO GET THESE MACDRAW GENERATED 'PICT's into the resource fork of the file I'm saving them in, so I can DeRez them in MPW.

From: the_cloud (Ken McLeod)
Subject: PICT moving

When all else fails, the Scrapbook desk accessory is a tried and true method of moving PICTS around. Try copying your MacDraw PICT and pasting it into the Scrapbook. Then go into ResEdit and open up the file you want to save the PICT into. Create a new PICT (select New, choose PICT from the scrolling list, and then select New again). Bring up the Scrapbook and select Paste! -k

From: andyc (Andy Cohen)
Subject: Hyperproblems

The discussions on the problems of building XCMDs and XCMDs over on the programming board have given some insight to problems in plain Hypertalking. Ever try to get the polygon tool to draw a polygon via "click at" or maybe try to get the freehand curve tool to drag to. Well, the poloygon tool ignores all click at commands (even though Hyper goes back to the browse command when told to) and the Curve tool draws only from one point to the next then closes the figure for each Drag. Hypercard only does things in a single minded serial manner. It can't do anything else until the called job is done then control is given back. The best approach then is to keep any functions to be called extremely brief in their execution time.

BTW using card buttons (set the LOC property) one can actually do some slick animations. A hard to find fact that is easy to deduce; just as in all things in the Mac environment fonts, picts and icon resources can reside in files as well as apps. If you are producing a Hypercard stack which MUST have a certain font and size or requires customized icons use resedit and paste these resources into the stack. If the stuff is already there it just duplicates them while the stack is opened by a Hypercard which already has them. If you do this then your stack can be used by any Hypercard.

From: rsobel (Richard Sobel)
Subject: 68010 PROCESSOR

Can anyone out there give the pros or cons for switching the 68000 chip with a 68010. ie. speed, extra instructions, etc

From: frank (Frank Henriquez)
Subject: 68010

The 68010 is pin and instruction compatible with the 68000. The main difference is that the 68010 can support virtual memory and memory management chips better than the 68000. The added instructions are mainly to deal with these new features. Some of the looping instructions have been modified, so that short loops are executed from the instruction cache, without having to go out and fetch it each time throught the loop. This speeds things up for very small loops. Some other instructions may have been tightened up a bit too. As to replacing the 68000 with a 68010: Motorola claims up to a 30% speed improvement. It's probably not much more than 10% for average operations. Since replacing the 68000 on a Mac means some *considerable* unsoldering (ever try to unsolder a 64 pin IC from a four layer PC board with really teeny traces?...) I don't think it's worth doing. If the 68000 were in a socket, then it might be worth the premium price of the '010.

Volume 4 Number 7

From: the_cloud (Ken McLeod)
Subject: TearOffPalettes Error

The TearOffPalettes source, as published in the April 1988 issue of *MacTutor*, contains an error which doesn't exist in the fully functional "Source Code Disk #31" version. Of course, there weren't any errors introduced into the PaletteWDEF.c file, since it wasn't even printed. In the TearOffMDEF.c file (the menu definition), the function ChooseItem() includes the following line:

```
if(!(tearOffMGlobals)->itemHilited)  (
    hiliteItem = (tearOffMGlobals)->currentItem;  /* wrong! */
(tearOffMGlobals)->itemHilited = true;        )
```

This is NOT correct, and should instead be:

```
 if(!(tearOffMGlobals)->itemHilited)  (
    saveItem = (tearOffMGlobals)->currentItem;  /* right! */
(tearOffMGlobals)->itemHilited = true;        )
```

as it is in the Source Code Disk version, else the hiliting gets screwed up real good.

From: lsr (Larry Rosenstein)
Subject: MF Background Bug

I just got the May *MacTutor* and on page 12 is a question about a possible MultiFinder bug. This bug is described in Tech Note #177. The letter had the bug almost right. It doesn't depend on the application having no windows. The bug shows up if:

(1) an application has the background bit set and calls WaitNextEvent with a sleep value > 50 and

(2) the application becomes the frontmost application because another application quits.

The maximum sleep value is dependent on the machine; larger values could be used on a Mac Plus and Mac SE. Also, condition 2 can be satisfied by running and then closing a DA. The bug is fixed in the upcoming 6.0 system. You can tell if the fix is in place by using the SysEnvirons call and checking the system version. (The Tech Note gives the wrong number for the check; the version should be greater than $0430.) My TimeKeeper program does this check and when running under System 6.0 will sleep for up to 1 minute under MF. (BTW, it also utilizes the Sound Manager on the Plus/SE under 6.0.)

From: davidw (David Whiteman)
Subject: Technote/Claris Question

Does anyone know whether the old Apple Technotes describing the file formats of MacWrite, MacPaint and MacDraw documents are accurate for the recent versions of the applications released by Claris?

From: rick (Rick Boarman)
Subject: Re: Technote/Claris Question

As far as I know the MacWrite 5.0 file format is the same as 4.6. MacDraw II has a completely different structure than v1.9. You can still save a drawing as a standard PICT though.

From: davidw (David Whiteman)
Subject: Re: Technote/Claris Question

Greetings, Claris finally returned my phone call: Basically with MacPaint and MacDraw, the two new versions have an entirely different document format, but if you create a document following the guidelines as described in the Apple technotes the new programs can still read it. With MacWrite 5.0 there are some subtle changes in the document format. Claris will not make the specifications for the newer versions public, nor post them, but if you write, and have a good reason for asking for them and sign an NDA, they will furnish it. *[So much for document format standards! Neither the new MacWrite or Word 3.01 formats are*

now easily accessable to developers. -Ed]

From: davidw (David Whiteman)

Subject: Mac II ADB problems

Some time ago I posted asking for information about problems with the ADB port on the Mac II.

After talking to some hardware experts I discovered the following: Apparently if you disconnnect anything from the ADB port while the Mac is powered on you may briefly short some pins. This causes a fuse to blow on the Mac II motherboard. Which would eventually mean that you would have to buy a new motherboard as the fuse is soldered in. The whole point is that if you are going to connect or disconnect the mouse or the keyboard power off the Mac first.

From: rdclark (Richard Clark)

Subject: Re: Mac II ADB problems

There's another reason to power down before changing ADB devices. Each ADB device has a microprocessor in it, and this mpu receives a "reset" command from the Mac at powerup. If you connect another ADB device in after the machine has been powered up (or disconnect and reconnect an existing device), the new device will not be reset properly.

From: wbigelow (Wendell Bigelow)

Subject: Missing Application Icon

I've created an application and added my own Application icon and icon for the files it will create. I do not see the fancy icon in the Finder when I view by icon, just the default application icon. I've read several places about removing the "Deskktop" file and otherwise forcing FINDER to rebuild the desktop and go out amongst ye Applications and re-build the BNDL resources. This worked for a previous version of the application (and the files it had created) that uses the same icon image but has a different resource type (4 character doo-dah). I've deleted that particular BNDL resource and even managed to set the "bundle" bit in the GET INFO dialog in ResEdit for my application and forced desktop rebuild....ALAS...to no avail... *[The best solution is to use a series of similar creator tags and to change the creator tag with each new icon you wish to use with a version of the program. Then when the bundle bit is set, the Finder will install the new icon with the new creator tag. However, if the tag has been previously used, you have to make the Desktop file "forget" the old icon associated with that creator tag. This is very difficult. It is much easier to teach the Desktop about a new creator tag and icon than to make it forget the icon associated with the old tag. -Ed]*

From: the_cloud (Ken McLeod)

Subject: Finder BNDL checklist

Did you: Make sure your application ICN# (and any ICN#s for documents created by your application) have corresponding FREF's?

Make sure there's a corresponding resource with ID=0 for whatever your application's creator type is? Make sure the BNDL's owner is the same as this creator type? Rebuild the desktop by holding down the Option and Command keys when quitting to the Finder? *[Be careful of this step; sometimes things get messed up, especially if you have a very full hard disk. -Ed]*

From: davidf (David Fandel)

Subject: List Manager/MacApp problem

Are there any MacApp experts out there? I am working on a list manager object that is a descendent of DialogItem. I have gotten the modal dialog box routines to work correctly. (ie. the scrollbars work, items are selected, etc.) But when I call the same routines from a modeless dialog routine the scrollbars do not show up initially. If the list is selected the paging regions and the thumb appear and move, but they can't be selected. The up and down arrows never show up. And if another window hides them they are not updated when the other window is moved. My call to this routine is based on a slightly modified DemoDialogs modeless dialog call. My code is as follows:

```
New(aDialogView);
FailNil(aDialogView);
aDialogView.ILMDialogView(    );
    {override - calls IDialogView}
aWindow := aDialogView.MakeOwnWindow(FALSE);
    {creates window}
IF aWindow <> NIL then
    begin
        aWindow.fFreeOnClosing := TRUE;
        aWindow.fOpenInitiallly := FALSE;
        SimpleStagger(    );
        aListView := aDialogView.DefineSimpleList(4,
18,TRUE,FALSE,theString);

{this is the call that places the List listed in theString
into item number 4 of the Dialog Box in an 18 pixel high cell.
The TRUE means a rectangle is drawn around the list and the
FALSE means the list is NOT alphabetized}

        aWindow.open;
    end;
```

The DefineSimpleList procedure runs new and FailNil on TSimpleList (a DialogItem descendent) and then calls the ISimpleList initialization routine which basically makes the list and attaches it to the DialogPtr after coercing it to a WindowPtr. Any clues on why the scrollbars go out to lunch when it is modeless? Have I forgotten an important call?

From: dcamp (David Camp, El Paso, TX)

Subject: cdev's

Hi! Can somebody show me the link command required to link the sample cdev in inside Mac V? Or the link command for the ApplFont cdev in MacTutor V3#10? This is for MPW Pascal by the way... I can't seem to nail it down and several people on GEnie can't either. *[All you need is a code resource. Do you need a special link command to get the MPW linker to create code resources? -Ed]*

From: the_cloud (Ken McLeod)

Subject: StartSound problem

I recently used HeapShow to check out a program I'm writing, and discovered to my chagrin that a handle wasn't getting unlocked, causing heap fragmentation. I tracked the problem down to this bit of code, which involves calling StartSound asynchronously:

```
HLock(handle(TheSound));
TheSound^^.FFSound.count := fixratio(1, TheSound^^.rate);
WITH TheSound^^ DO
    BEGIN
```

```
        StartSound(@FFSound, WaveLen, pointer(Syn));
    END;
HUnlock(handle(TheSound));
```

When (Syn) is -1, the sound is produced synchronously; however, when (Syn) is 0 and the sound is being produced asynchronously, the handle never gets unlocked. In fact, my calls to HLock and HUnlock the handle appear to have no effect... I removed them, and the handle was STILL locked on exit from the routine. Does anyone know if there's something I'm doing wrong here, or if there's a problem with StartSound? I seem to recall some thread about StartSound here before... I should mention that my handle to the sound is declared as a global variable... is that a factor?

From: lsr (Larry Rosenstein)
Subject: Re: StartSound problem

There is a general problem with your code as written, which could manifest itself as a locked handle (but I don't know for sure). The problem is that StartSound expects a pointer to the sound data. Locking the handle is the right thing to do, except that it has to be locked for the duration of the sound. If you call StartSOund synchronously, then there is no problem. If you call it asynchronously, then you can't unlock the handle until the sound finishes. The correct thing to do is to use a completion routine to detect when the sound is finished, set a flag, and unlock the handle when your program sees the flag set. You can't unlock the handle in the completion routine, because it runs at interrupt level, and could have interrupted the Memory Manager. (If you use a global flag, you also have to be careful to setup A5 correctly, and watch out for MultiFinder.) I don't know if this could cause the handle to stay locked, but it could certainly cause problems playing out the sound if you unlock the handle while the sound is playing.

From: the_cloud (Ken McLeod)
Subject: More StartSound...

Well, I tried setting a flag after I called StartSound asynchronously (passing nil for a completion routine), and putting a few lines of code in my main event loop to unlock the handle if SoundDone was true and the flag was set. It should have worked...but didn't. Then I got an inspiration, and thought of StopSound; so... this works:

```
if (unlockMeFlag and SoundDone)  then
    begin
        StopSound;
        HUnlock(handle(mySoundHdl));
        unlockMeFlag := FALSE;
    end;
```

But without the StopSound call, the handle remains locked! If we get past SoundDone there should be no need to call StopSound; this is truly bizarre. StopSound "executes the sound's completion routine" as well as cancelling any pending calls (there aren't any), so perhaps it won't allow the handle to be unlocked if a sound is played asynch until some sort of completion routine is called? I dunno... anyway, StopSound did the trick.

From: jimr (Jim Reekes)
Subject: Re: More StartSound...

The glue code in MPW for StopSound unlocks the handle created by StartSound. StopSound glue checks for the handle and will unlock it for you. The meaning behind all of this is to call StopSound before ExitToShell.)]> reekes <[(

From: tony (Tony Fashoro, San Jose, CA)
Subject: Calling Code Resources

Help, I want to write a standalone code resource using LS Pascal. I need to know how to call the code resource form within an application. I beleive I have to use the resource manager to load and lock the resource and then do something to the handle returned by the resource manager. What is that something ?

From: think (Richard Siegel, Bedford, MA)
Subject: Re: Calling Code Resources

Calling the code resource is pretty straightforward:

```
procedure CallCode(arg1, arg2,...,argN, codeH);
INLINE $205F, $2050, $4E90;
```

The first N arguments are the arguments that your code resource's MAIN routine expects, and "codeH" is a handle to your code resource.

NOTE: Before calling the above procedure, you should either lock the code resource (you can set its locked bit in the resource attributes, if you want) or make provision for it to lock itself. If the code resource moves while it's executing, you're dead... —Rich Siegel, THINK Technologies

From: polaris (Jamal Hannan)
Subject: Resource Problems

Hello. I need a little help here... I am writing a program in Pascal & one of the procedures in it needs to get hex data from a resource of type DATA into an array, where each byte can be utilized individualy... I can't seem to get my program to work.. here it is:

```
PROGRAM Arry;
TYPE
    myArray = ARRAY[1..10] OF INTEGER;
    myArrayPtr = ^myArray;
    myArrayHdl = ^myArrayPtr;
VAR
    filename : STR255;
    theResID : INTEGER;
    theRefNum : INTEGER;
    anArrayHdl : myArrayHdl;

PROCEDURE QuickAndDirty;
    BEGIN
        theRefNum := OpenResFile(filename);
        anArrayHdl := myArrayHdl(Handle(GetResource('DATA',
theResID)));
    END; { QuickAndDirty }

BEGIN
    filename := 'Rscs';
    theResID := 1000;
    QuickAndDirty;
END.
```

Anyone know whats wrong? Can you show me a routine that would work?

From: thecloud (Ken Mcleod, La Habra, CA)
Subject: Re: Resource Problems

Is it possible that GetResource('DATA', theID) is conflicting with a DATA resource in another open resource file (like the

compiler, or the System?) Perhaps using Get1Resource instead of GetResource might help... (just a guess, nothing more).

From: lsr (Larry Rosenstein, Cupertino, CA)
Subject: Re: Resource Problems

It is possible to get a resource conflict. In your application, however, your application resource file, or any resource file that you open will be searched first. Only if it is not found there will other opened resource files be searched. (Also, under MultiFinder, other applications won't be visible at all.) Get1Resource might solve the problem. Also, you can see if this is in fact the problem by calling HomeResFile on the resource you get and see if it coming out of the resource file you expect.

From: polaris
Subject: resource

Help... It was written in Lightspeed Pascal & I had the linker automatically include the resource file so I didn't need to call GetResource Anyway... that code WOULDNT work and simply by looking at it probaly wouldn't tell anyone much.... anyway, I wrote some compleatly different code and it worked... here it is:

```
PROGRAM Open; {Jamal Hannan ... 5/18/88}
VAR
    thing : PACKED ARRAY[1..200] OF INTEGER;
        {setup addressable array block}
    resType : STR255;
    resID : INTEGER;
    resHandle : Handle;
    resPtr, thingPtr : Ptr;
    blockSize : Size;
BEGIN
    resType := 'DATA';     {load up the resource...}
    resID := 1000;
    resHandle := GetResource(resType, resID);
    LoadResource(resHandle);
    resPtr := resHandle^;
        {get resource pointer from handle}
    thingPtr := @thing;
        {get pointer to point to array block}
    blockSize := GetHandleSize(resHandle);
        {get size of data block}
    BlockMove(resPtr, thingPtr, blockSize);
        {move res data into array space}
END.
```

The previous code was really something mostly written by someone on EchoMac (sort of a FIDO or OPUS email system for Macs)... it really didn't work for me so I wrote my own. I think the problem with the last section of code was some Ptr or Handle type coersion... some pointers ended up pointing to the wrong memory blocks... NOW what I need is a bit of code that will save an ARRAY into a RESOURCE.. exactly the opposite of the above... but I am having alot of trouble... can anyone help me with that?

From: rdclark (Richard Clark, Tustin, CA)
Subject: Saving an array to a resource

Well, if what you want to do is save an array intact, and restore the exact same array to the exact same state, you can do the following: (* DANGER! DANGER! DANGER! This code has not been tested! *)

```
procedure DumpArray(theArray: arraytype; theRes: resType;
resNum: integer);
var
    scratchHand: Handle;
    thisResFile: integer;
    done : boolean;
begin
{If an old copy of the resource exists, remove it }
thisResFile := CurResFile;
{Should be done in main program, since you want the resource
number of your application. }
done := false;
repeat
    scratchHand := GetResource(theRes, resNum);
    if (scratchHand <> nil) then
        if (HomeResFile(scratchhand) <> (thisResFile)) then
            done := true;
        else
{ We have one or more copies of this same resource in our
file, so delete it (and dispose of the body) }
            ReleaseResource(scratchHand);
until done;

    { Now, we can create a handle to hold the array }
scratchHand := Newhandle(sizeof(arraytype));
    { Copy the data from the array to the resource }
BlockMove(@theArray, scratchHand^, sizeof(arraytype));
    { Now, put the resource into your resource file }
AddResource(scratchHand, theRes, resNum, '');
ChangedResource(scratchHandle);
    { Checks to be sure there's disk space }
WriteResource(scratchHand); {Save it }
end;
```

From: rguerra (Rich Guerra)
Subject: Multifinder questions

I have a couple of questions regarding programming for Multifinder.

1) I'm trying to write an application that will be able to sublaunch another application while running under MF, but does not itself close. Control is transferred to the newly launched application which is now in the foremost layer. I've tried the code in Tech Note 126 (Sublaunching) and set the two high bits of LaunchFlags as specified in an addendum to the Multifinder developer's documentation. However, the application gets launched and my application quits. Has anyone tried this code and made it work?

2) How does one determine how much memory is a) allocated for an application under MultiFinder and b) how do you determine the amount of the allocated space that is actually used. (i.e., something similar to the Finder About box that gives all this information graphically) Any suggestions would e appreciated.

From: thotpolc (Bill Evans, Irvine, CA)
Subject: Complaint Time

Okay, fellas it's Complaint Department time.

Complaint One:

In the MacsBug documentation there is mention of something called the "current heap zone" under the HX, HC, and HD commands, and by implication also the HS and HT commands. This is completely different from the "current heap zone" as discussed in Inside Macintosh and determined by the TheZone global variable (at $118). It's quite good that they're different; it gives one a certain debugging flexibility. But (at least in MPW 1.0) the MacsBug documentation doesn't say that they're different. So when I forgot to change TheZone back to the application heap, I didn't suspect it at first because the HC command quite

innocently told me that my current heap was still the application heap. So beware. Check $118 if you want to know whether your program is working in the right heap zone.

Complaint Two:

The Inside Macintosh documentation for Secs2Date (p. II-380) says that it returns a pointer to a date/time record. Wrong. You set up the pointer in A0 to a date/time record for which you have allocated the space, and Secs2Date will fill that record. One would expect IM to be more clear on this point, since in most cases it is scrupulously careful in questions of this sort.

Complaint Three:

I'm sure most people who are interested know this, but I'll state it for the record. The Inside Macintosh documentation for GetHandleSize (p. II-33) says "in case of an error, GetHandleSize returns 0". Wrong. In case of an error, GetHandleSize returns the (less-than-zero) error codes listed at the bottom of the page. But those result codes aren't quite right, either. They show noErr as meaning "No error". Actually, in case of no error, you get the (greater than or equal to zero) size.

Complaint Four:

This last one isn't about documentation; it's about a botch in design details of QuickDraw. When you use the Memory Manager to get and resize handles and pointers and the like, you have adequate warning when an operation fails because of lack of memory. You can go off and pursue an alternative course of action without bombing the system. Not so when QuickDraw causes you to run out of memory. The best you can do is make the current heap be something whose grow zone function at least warns the user that he's run out of memory — but even this function, returning "failure" to QuickDraw, can't prevent QuickDraw from bombing "25". If we're encouraged (and rightly so) to be fastidious about checking for no more memory with Memory Management traps, why can't we be given the same opportunity with QuickDraw? Granted, a region may be fouled up beyond repair if QuickDraw runs out of main memory, but can't it just then set it to contain nothing and let us decide what to do from there? *[I'm convinced no one at Apple understands the proper way to do memory management on the Mac or they would have told us by now. It is the number one "black magic" art of Macintosh Programming. Scott Knaster's books made the best effort at the subject, but that only scratched the surface of what is needed. -Ed]*

From: cmkrnl (Eric Slosser)

Subject: recover from a HD crash

The following program is meant to read the last block of SCSI drive 0 and copy it to the 2nd block. The purpose is to overwrite a corrupted Master Directory Block so that Disk First Aid will be able to repair the disk. (As outlined in Tech Note #134, p.11) This is my first attempt at using PB driver calls, so I've messed something up. I get an 'Illegal Err' during the PBRead. The PC is somewhere in 'Create' (according to Macsbug). Anyone care to comment (once you stop laughing) on this?

```
PROGRAM FixDisk;
USES {$LOAD PasSymDump} Memtypes, Quickdraw, OSIntf, ToolIntf,
PackIntf;
CONST
    SCSI_Zero_dQRefNum = -33;  { IM IV-215 and IM II-191 }
VAR
    theQHdr : QHdrPtr;   { IM II-373 }
    aQElem  : QElemPtr;  { IM IV-181 }
    thePB   : ParamBlockRec;
    theBuffer: ARRAY [1..128] OF INTEGER; { total size = 512 }
    SizeOfVol: LONGINT;
     err     : OSErr;
BEGIN
    theQHdr := GetDrvQHdr;
    aQElem := theQHdr^.qHead;
    debugger;
        { we are going to babysit this code }
        { Go through queue, looking for SCSI device ZERO }
    WHILE aQElem <> theQHdr^.qTail DO
        {we do this to get the ioVRefNum}
        IF (aQElem^.DrvQElem.dQRefnum <> SCSI_Zero_dQRefNum)
THEN
            aQElem := aQElem^.DrvQElem.qLink;
            IF (aQElem^.DrvQElem.dQRefnum =
SCSI_Zero_dQRefNum) THEN
            BEGIN
            { It's an 80MB drive, so we have to combine 2
fields for the size }
            SizeofVol := BSL( aQElem^.DrvQElem.dQDrvSz2, 16 )
+ aQElem^.DrvQElem.dQDrvSz;
        WITH thePB DO
            BEGIN
                ioCompletion := NIL;
                ioVRefNum := aQElem^.DrvQElem.dQDrive;
                ioRefNum := SCSI_Zero_dQRefNum;
                ioBuffer := @theBuffer;
                ioReqCount := 512;
                ioPosMode := fsFromStart;
                ioPosOffset := (SizeofVol -1)*512;
            END; { of filling in the param block }
        err := PBRead( @thePB, FALSE );
        { we die inside this call. Why?}
        IF (err<>noERR) THEN
            debugger;
        { ... the code would go on here to mimic the above,
only writing }
    END;
END.
```

Volume 4 Number 8

From: pepnerd (Peter Kramer)

Subject: menus

I'm just a beginner at programming the mac. I'm trying to set up menus using the code in *"Using the Macintosh Toolbox with C"* by Takatsuka et. al. I'm not sure how to call InitGraf() in the situation where I have not set up any windows, only menus. In my development system the example (Aztec C) the call to InitGraf() uses the argument &thePort. However I cannot find where thePort is defined. Its quite distressing to double click on an application icon and get nothing but a bomb!

From: ericlim (Eric Lim, Flushing, NY)

Subject: Re: menus

I'm not familiar with Aztec because I'm using MPW C 2.0.2 and I think the variable thePort is defined in the "Quickdraw.h" header or include file.

From: think

Subject: Re: menus

It's not important to know where thePort is defined; all that's

important is that you pass the correct series of initializations in order to get your application started:

```
{
InitGraf(&thePort);
InitFonts();
InitWindows();
InitMenus();
TEInit()
InitDialogs(NIL); }
```

Typically, thePort is defined as an extern GrafPtr in Quickdraw.h (in LightspeedC) or in the appropriate header for other C compilers. —Rich Siegel, THINK Technologies

[The variable thePort is one of those mysterious Quickdraw global variables that no one ever talks about once you are convinced the proper call works correctly. The first few issues of MacTutor, over three years ago, addressed this subject in depth, explaining how the Quickdraw Globals are stored relative to A5 and how they work. The Mac system is responsible for the Quickdraw Globals maintenance. Refer to Volume 1 of The Best of MacTutor for more information. -Ed]

Subject: cdev's and System 6.0

Practically all the public domain/shareware cdevs I used with no problem under System 4.2/Control Panel 3.2 appear to break under System 6.0/CP 3.3. Usually it's "The Control Panel cannot get enough memory" (with almost 2 megs free?!), but other alerts have popped up as well. One of the "broken" cdevs includes the Apple "Sample" cdev from IM V, which I compiled myself. Hmmmm.... I think the problem may be that these cdev's do not have a 'sysz' resource. Am I right?

Well, after looking more closely, I discovered the reason why various cdev's were breaking under System 6.0: they check for the presence of certain packages in the System file (usually PACK 7 or PACK 4) that no longer exist! (except in the ROM). Even Apple's Sample cdev in IM V does this (although there's a brief disclaimer that you "really should" check for the packages in ROM). Ouch. Then there's the MultiFinder test. Most sample source code I've seen (actually, ALL) checks for MultiFinder's presence by testing to see whether the WaitNextEvent trap is implemented. Now, in System 6.0, it's implemented ALL THE TIME, whether you're in MF or not. So if application X assumes it's running under MultiFinder when it isn't, well, look out.

Question: how do you detect MultiFinder's presence or absence under 6.0? *[Answer: Your not supposed to care if MulitFinder is there or not; only check for the various services you expect. -Ed]*

From: tomh (Tom Herbst, Ithaca, NY)

Subject: Palette Manager in LSP

I am having a great deal of difficulty adding the necessary interface file to use the Palette Manager with LightSpeed Pascal. I manually entered the appropriate data structures, inlines, etc. gleaned from Inside Mac V, the MacTutor articles, and the Programmer's Online Companion (all consistent), but the SetPalette calls crash even MacsBug with bus errors. Is there a copy of the necessary code along with a simple (i.e. create a palette, a NewCWindow, and draw a rectangle) program example out there somewhere? My PaletteMgr and example program compile ok, but run very inconsistently, sometimes drawing things in the default (startup) palette and more frequently crashing the whole shebang.

From: ericlim (Eric Lim, Flushing, NY)

Subject: Modeless Dialog/List Manager

HELP! HELP!! Does anyone know of a technique on using the List Manager Package in a UserItem inside a Modeless Dialog? I don't want to create the window and controls on my own, I want to use IsDialogEvent and DialogSelect. Any suggestions on how I can do this? So far, I'm getting a system error when I use LNEW with rView set to the rectangle of the UserItem. I got the Modeless Dialog to work on a regular dialog with a UserItem. I also got the List Manager to work on a regular window that I created. But it just bombs when I use Modeless Dialog with the List Manager Package. I would appreciate any help or suggestions I could get. Thanks a lot! *[Forget the Dialog Manager and just use the list manager into a window. -Ed]*

From: tompink (Tom Pinkerton, Iowa City, IA)

Subject: Ending DA drvrCtl routine

For a while now, I've been using an ending to the drvrCtl desk accessory routine that was supplied with my MDS development package. That ending, rather than simply returning, calls the low-level Device Manager routine IODone (pointed to by the JIODone global variable) before returning. I've been blindly keeping this in all my DA sources without really knowing its purpose. I assume that who ever wrote the sample DA for the MDS development system knew what he or she were doing when they added this call. All I can figure is that the call somehow clears the IO Request sent to the DA from the driver IO Queue so that other devices won't receive the request (since they needn't bother with it, I guess). Does anyone know anything about this?

Volume 4 Number 9

From: bruceh (Bruce Henderson, Canoga Park, CA) Sat, 2 Jul 88 15:22:46 EDT Comment: further comments

Subject: MAC and MC88000

The Motorola 88000 is pretty much THE processor of the future. After going to the Motorola seminars and studying the docs on it, I feel that the power offered here is something that hopefully Apple won't pass up.

I think the optimum design would be to use to 88100s. One for graphics only and the other for everything else. If both of the processor modules used 2 88200 CMMU's on the instruction side of the path (32K) then it would be possible to store 1 segment in the CMMU at all times. [more on why this is importiant later] Other system memory could be configured to operate in burst mode. So as the 88100 needed another segment, it could read it in a "BURST". [for those of you not so hardware intensive hacks burst mode allows the CPU to fetch a large amount of data from memory at the highest possible rate]. The 2 processor sets could be connected via the 88000 p bus so that the interrupts could

occur 1 cycle. Yes, this idea doesn't have a lot of refinement, but I have been doing a great deal of thinking about ways that a 88K Mac could exist, achitecture and such. And I have decided to appoint myself the independent Mac88000 evangelist. The power is just too great to pass up.

Wait, you say. The 88000 isn't software compatible with the 68000! Ah, this is where the ROM/Toolbox guys at Apple will have to earn thier pay. The trick is to make the Mac88000 source code compatable with the rest of the Mac line. If every last one of the Mac ROM calls can be implemented on the Mac88000, then the trick is just to allow MPW to generate 88000 code as well as 68000 code. So to make an Mac88000 version of a current peice of software would only requore a recompile with the {+88000} flag set. This may seem like too much to ask for, but the truth of the matter is is that if Apple were to go ahead with this now, while the hardware hacks were making the box, the toolbox critters could be already creating the code for this things ROMS, thanks to the Tektronix 88000 MacII board. So really, the ROM development team doesn't have to wait until the hardware guys had somthing that works. So I know that all of you LightSpeed guys are howling out there. I think that Apple should make sure that the major compiler writers are seeded as soon as they have anything that works!

Well, there is a lot more to my Ideas, If anyone is really interested they can contact me, or just wait for the next time I feel like having an 88000 revival meeting.....

From: frank (Frank Henriquez, La, CA)
Comment: to #81 (bruceh), further comments
Subject: Re: MAC and MC88000

Yeah, the 88000 is real swell, but it's Motorola's first attempt at a RISC machine, and it does have it's nasties: Forget about writing code in assembly language. Not only is the instruction set (and addressing mode) extremely limited (well, it's a RISC machine) you have to keep track of azillion little CPU states and modes and whatever. So, we've settled on writing in a HLL. Fine. To really squeeze out the maximum performance out of the beast, the compiler has to be smart enough to know when and how to generate code that executes in parallel. I don't think we'll be seeing LightSpeed C (or any other compiler) for the 88000 *that is any good* selling for peanuts...or a nice pile of C notes. The hardware is a dog. Too many things going on at once. The bus interface to the "rest of the world" (memory, peripherals) is a multiplexed 32 bit bus with some unfriendly bus signals (WHAT! no Read/Write signal? sheesh) Yeah, it'll take a while to get used to the beast.

I'd rather see a 68030 (or 68040) running at 88000 speeds with some intelligent memory controllers (to eliminate or minimize the need for an obscene # of wait states) Motorola has promised such a beast (the 68040) at some point, and it would be compatible with 68000 and 68020 code.

Of course, you could emulate the 68000 with an ECL version of the 88000, but I don't want to buy a liquid nitrogen cooling system for such a machine...

The 88000 is a great machine, and one notable feature is that the microcode can be customized. I'd love to see an 88000 based graphics controller for the Mac. Quickdraw (or Display Postscript, or whatever) would really scream.

I wouldn't mind a 50 MHZ 68040 Mac with 1G or ram and a terabyte optical HD and a 2000 x 2000 pixel 48 bit color display with an 88000 for the graphics. Who needs a Cray, anyway?

From: rusty (Mr. Rusty Hodge, Orange, CA)
Subject: Re: MAC and MC88000

The 88000 is more comparable to the AMD 29000 than to the 68000 family. If you treat the 88000 like a microcoded HLL engine, I think it will perform nicely. I think that Apple will end up with even a more custome machine than the Mac when (if) they go to the 88000.

If the 88000 is microcoded properly; it will be able to give the performance to the kinds of tasks that the Mac OS need the most performance in. However Apple is going to have to create the microcode to turn the 88000 into a usable HLL engine.

Maybe the 88000 will become the basis for a true Hardware-Quickdraw? Or maybe the basis for an PostScript HLL engine? It isn't that hard to do.

Volume 10 Number 10

From: rkk (Russell Kadota, Huntington Beach, CA)
Subject: menus & resedit

I'm just getting started in Mac programming, and I've run across something that one of you more experienced folks might be able to explain. I was writing a simple program to just put up a menu bar and handle DAs. I created a resource file with the appropriate MENU resources using Resedit 1.1b3. After much debugging anguish, I found that the MENU resources didn't have the resource numbers stored in them. I had changed them to 128, 129, etc. using the 'Get Info' command in Resedit. Is this a problem with Resedit, or with the MENU TMPL, or am I doing something wrong? I'm using LightspeedC, on a vanilla mac plus. Thanks in advance.

From: thecloud (Ken Mcleod, La Habra, CA)
Subject: Re: menus & resedit

The function GetMenu() takes one argument: the resource ID of the MENU resource you've created. This is the same number you've set by using ResEdit's "Get Info" command ("128, 129, etc."). You don't say what problem you experienced, but if you're using LightspeedC, the MENU resources should be stored in a file named <PROJECTNAME>.rsrc, where <PROJECTNAME> is the name of your Project file. The individual resources don't have the resource ID numbers stored *in* them, but the toolbox routines you'll call to get and make use of the menus (or any other resource) don't care; just pass them the ID number, and assuming a resource of type MENU with that ID exists, the function will return a handle to it! You can then pass that handle to other functions that need a MenuHandle as an argument... If you can't solve the problem, explain what you're trying to do in more detail and hopefully we can help...

From: rkk (Russell Kadota, Huntington Beach, CA)

Hmm, I guess I wasn't very clear in my description. I created the resource file (named appropriately), and set the resource ids of the menu resources to 128, 129, and 130 (apple, file, and edit). The program finds the resources just fine, and draws the menus and all the items correctly. But when I try to select something, MenuSelect returns a 0 for the menu id (although the item number is correct).

What made me think that the menu id (same as resource id?) was stored with the resource was that I used Fedit to inspect some other resource files and found a number which matched the resource number 14 bytes before the first character of the first item, in each case. So I used Fedit to put a similar number in my resource file, and presto, everything works. So, I guess the problem is "solved", but not too elegantly.

From: thecloud (Ken Mcleod, La Habra, CA)

MenuSelect returns a long... the hi word should contain the menu ID, and the lo word contains the number of the selected menu item. You would normally do something like this:

```
DoMenuEvent(MenuSelect(Event.where)); /* MenuSelect result */
...
...
void DoMenuEvent(menuChoice)
long menuChoice;
{
  short whichMenu;
  short whichItem;
    whichMenu = menuChoice >> 16;   /* HiWord(menuChoice). */
    whichItem = menuChoice & 0xFFFF;  /* LoWord(menuChoice).
*/

/* use switch(whichMenu) to go to other functions for
   handling each menu, passing them (whichItem). */
...
}
```

Now, assuming you're doing this, and still getting 0 in (whichMenu), possibly the MENU resource wasn't written properly when it was created or changed. I looked at some MENU resources with ResEdit's "Open General", and the first word is indeed the resource ID (0080 for a menu with ID=128,and so forth). You shouldn't have to use Fedit, though, or do any kind of bit twiddling with the resource to make this happen! You might want to try using a different version of ResEdit (I've been using 1.2d1 with no problems; 1.2d2 is the latest I've heard about), or compiling your menus with RMaker, to see if you can isolate your copy of ResEdit as the problem.

From: rkk (Russell Kadota, Huntington Beach, CA)

Thanks for the help. It turns out that it was indeed my copy of ResEdit. I picked up a newer version (1.2d(something)) and everything works OK. Thanks again!

Volume 4 Number 11

From: rick
(Rick Boarman, Claris Corporation)
Subject: MacDraw II PICT2 Spec

The file format spec for MacDraw II is now available!! To get a copy of it all you need to do is write a letter letting us know why you want it and what you're planning. We'll approve all reasonable requests. Send the letter to:

Claris
440 Clyde
Mountain View, CA 94043
Attn: MacDrawII Product Manager

From: emmayche
(Mark Hartman, Fullerton, CA)
Subject: System 6.0.1

Well, the unofficial distribution is finally here - and seems to be OK. None of the color problems we had with 6.0, and fixes to some we had in 4..2. I'll let everybody know if we run into any problems.

I'll be making the rounds of the user group meetings in Orange County for redistribution purposes (and promotional purposes - Photon Paint is now at your local dealer).

Get System 6.02, NOT 6.01!
From: rdclark (Richard Clark, Tustin, CA)

Is NOT the last word in Systems. (Just read your AppleLink, Mr. Developer) Apple just warned all of the Certified Developers NOT to ship 6.0.1 with their products, but to get System 6.0.2, which is due out shortly (this week?)*[Richard is right. System 6.01 is buggy. The OFFICIAL release of system 6 is 6.02. -Ed]*

From: rguerra (Rich Guerra, Iowa City, IA)
Subject: MultiFinder Magic

It seems the humorous About box in MultiFinder 6.0 was hidden a bit better than in the first release. If left alone for about 1 hour, the 'About MultiFinder' dialog will flash a funny message. But for those of you who don't want to spend an hour staring at your screens, you might try this: Using a file editing program like FEdit, SUM, or even ResEdit search for the hex string 034BC0 and change to 0000C0. There are two occurrences of this string, so change both. If you're using ResEdit, open the CODE resource 1 ("Main") and use the Hex search function. This applies to MultiFinder in System Release 6.0. I don't know if this will work on the new 6.01 or higher. After this patch, the about dialog will alternate between the standard text and the funny message. Remember, DO THIS ON A COPY!!!!

From: miket
(Michael Twitty, Los Angeles, CA)
Subject: Apple Price Increases

There seems to be something I don't understand about this price increase. If their justification is the current high RAM prices, why did they raise the price of their monitors? Trust me, I looked inside...no RAM! *[The price increases across the board are considerable. Since the Macintosh was already overpriced to begin with, this just makes it worse. We welcome predictions on the market effect of this move. -Ed.]*

From: rusty (Mr. Rusty Hodge, Orange, CA)

Apple Corporate is getting fat, lazy, and inefficient. Look at

the System 6.0 fiasco. Why did they FedEx 6.0.1 to developers telling them that it was still buggy and to wait for 6.0.2? Apple is gloating in their success. They are convinced that they can do no wrong. They spend their money making their headquarters fun to work in for their employees. They keep the employee refrigerators stocked with beer and wine coolers.

It seems that most of the talent is gone from Apple. A few bright, hard working people are making all the progress for an otherwise stale company. Face it. These price increases are unfounded. They are just another creative/strange marketing idea: If they raise the price, more people will buy the machines fearing that the price will increase again! *[That only works for cars and other necessities. -Ed]*

From: emmayche
(Mark Hartman, Fullerton, CA)
Subject: Playing snd 2 resources

Here's a message to MACDTS from one of the other programmers here. Any ideas, anyone?

I've been through the IM-V chapter on the Sound Manager with a fine-toothed comb, and my supposedly simple quest to have my Mac II play a sound that's been digitized and is in a 'snd ' type 2 resource has me stumped.

Here's some code (Think-C) that doesn't work:

```
#include      "SoundMgr.h"
#define nil    0L
main()
    {
     Handle effect;
     SndChannelPtr channel;
     SndCommand sound;
     effect = GetResource('snd ', 10000);
     if (effect == nil)
          panic("can't GetResource");
     HLock(effect);
     channel = nil;
     if (SndNewChannel(&channel, 5, initSRate22k, nil) !=
noErr)
          panic("can't SndNewChannel");
      if (SndPlay(channel, effect, TRUE) != noErr)
          panic("can't SndPlay");
      sound.cmd = noteCmd;
      sound.param1 = 1000;
      sound.param2 = 0xFF000037;
      if (SndDoCommand(channel, &sound, TRUE) != noErr)
          panic("can't SndDoCommand");
      if (SndDisposeChannel(channel, FALSE) != noErr)
          panic("can't SndDisposeChannel");

      HUnlock(effect);
      ReleaseResource(effect);
    }
```

Now, my interpretation of IM-V is that I have the sound digitized in a snd-2 resource, and that I load it with SndPlay and then I play a note to get it to come out at whatever frequency and duration. However, it never gets that far. I always get a "can't SndPlay" output, with the error code of "badFormat" returned. I've single-stepped through the SndPlay trap, and what it "looks" like happens is that SndPlay looks at the 'snd' resource type and won't play anything but snd-1 resources. So, what's the story? How do I play a digitized sound, if I'm not supposed to use SndPlay?

I have a bunch of other Sound Manager questions that I'll defer for the moment. If anyone would like to volunteer as a Sound Manager expert, I'd appreciate the help. If anyone can send me some code that works with the Sound Manager, I'd appreciate looking at that as well. At this point I'm frustrated enough by all of the things that I *have* tried that I'll appreciate *anything*.

From: mkg (Marsh Gosnell, Montclair, NJ)

The following code plays both type 1 and type 2 snd resources.

```
theHandle = GetResource('snd ', id);
 LoadResource(theHandle);
 HNoPurge(theHandle);
 SndPlay(0, theHandle, false);
```

It works fine on my Mac II and Mac + both running 6.0

From: retzes
(Steven Retzlaff, La Mesa, CA)

I had a similar question in the programming/pascal topic, One problem seems to be the earlier system release: a bug in format-2 SndPlay was fixed in System 6.0.

From: willcox (Dave Willcox, Chicago, IL)
Subject: multifinder and clipboard

I am having a clipboard programming problem under Multifinder. Various programs (4th dimension, LabView, etc) allow for the creation of "external procedures" in Pascal, etc. I have written a procedure to put TEXT onto the clipboard so that it could be pasted into other programs under Multifinder. First, this is necessary because programs like 4D can have the user put a given field on the clipboard, but can not combine fields, etc into a single clipboard entity. The procedure is a simple two line program: ZeroScrap, and PutScrap. However, the following happens under Multifinder (5.0, 6.0, take your pick). First, my program works as evidenced by a "Show Clipboard" while still in 4D. However, after a context switch, the clipboard in, lets say, WriteNow is empty. If I copy something in WriteNow, it shows up in the 4D clipboard. Strangely, I can copy my stuff onto the ScrapBook under Multifinder, and then do a context switch into WriteNow, and copy it again out of the Scrapbook. However, this is a two step process for the user, which should be simpler.

I am convinced that this has to do with MultiFinder's use of the Scrap. Each application has its own scrap, which must be converted when a suspend event is received. However, 4D does not do this because it has no private scrap, so it thinks that it doesn't need to do a conversion. I have tried to change the MultiFinder flags in the SIZE resource, but 4D currently says that it does not accept suspend/resume events anyway.

From: kevink (Kevin Killion, Chicago, IL)

Don't know if it's related, but I'm wrestling with a nasty scrap problem also. In my app, if more than about 10K of material (TEXT, PICT or whatever) is placed on the scrap, there will be a crash when the program is Quit. It can be as simple as copying a largish item from the Scrapbook and then Quitting. As you said, it's a simple matter of ZeroScrap, PutScrap so it's hard to see what could go wrong. The crash only happens on a Mac II, and only if the scrap has more than 10K or so. According to TMON, the crash occurs at $AD92, which TMON labels as

GetNewPalette.

From: retzes
(Steven Retzlaff, La Mesa, CA)
Subject: Writing a Serial Driver

I am interested in writing a specialized Serial Driver to get more direct control over interrupts and the 8530-SCC registers. What I am wondering is how to deactivate and/or redirect certain system resources that interfere with serial communications such as: 1: The Existing ROM Serial Driver which the call to RAMSDOpen presumably closes. 2: The Disk-Driver feature which automatically transfers characters from the serial port to the system Serial-DRVR while interrupts are disabled during disk access(IM Volume 2: About the Disk Driver). I would appreciate any comments or suggestions, or any others regarding writing a serial driver.

From: rdclark (Richard Clark, Tustin, CA)

Unless you REALLY need that much control, writing your own serial driver isn't such a hot idea, since when Apple changes the HW, you'll probably break. *[See the December and January 1988 issues of MacTutor for an example of a printing driver and be prepared for a lot of work. -Ed]*

From: dsa (Dave Stine, Saugus, CA)

Are you sure that you want to invest the time & heartache necessary in writing a Macintosh driver? Mac device drivers, especially those intended to run under MultiFinder are not for the faint of heart. If you wish to supplant the serial driver that is called when your application performs a PBOpen() on .Ain, what you might consider doing is putting your DCtlEntry in the UnitTable where the DCtlEntries for the old .Ain and .AOut are. This way, all the Mac ROM calls to the supplied serial driver will now call your driver. As for how the .Sony driver grabs interrupts intended for the serial driver and makes the characters appear in the serial driver's buffers when the .Sony driver re-enables interrupts, I would suggest that you disassemble the .Sony driver and the serial driver. The interrupt conflict in the .Sony driver with the serial driver would most cleanly be solved by the use of DMA in reading the disks, but I guess Apple hasn't invented DMA yet.

Volume 4 Number 12

Because this is the international edition, we thought we' originally planned to translate the entire article into Japanese. However, since MacTutor doesn't have the Kanji fonts for the LaserWriter, it was decided just to pretend that the column was translated into Japanese and back to English again (like the rest of the magazine was!).

From: wayne (Wayne Correia, Cupertino, CA)
Subject: Mac IIx

The Mac //x is a 16 Mhz. 68030 with a 68882 math coprocessor. The new floppy drive and SWIM chip (Super Woz Integrated Machine) allows the reading and writing of Mac formatted 400k/800k/1.44mb disks, 800k Apple ProDos, 720k/1.44mb IBM formatted disks. The logic board is available as an upgrade for a little over $2000 and the drive and SWIM chip upgrade is available for $599. The 68030 has built-in memory management, so a 68851 PMMU is not required for A/UX operation. The base machine also comes with 4 Meg RAM on the motherboard, and room for 4 more. The ROMS are now mounted on an Apple ROM-SIMM board that plugs into a SIMM socket where the old 28 pin DIP ROMS used to be. Also, If you are only interested in the 1.44 drive, you can just buy the drive kit which installs in any Mac II. I have had one of these on my desk for about two months, and the //x is VERY compatible with Mac applications!

From: wildman (Randy Saunders, Upland, CA)
Subject: Getting Turned Down for developership

Yes, it does seem the rule is "Will you be selling something at ComputerLand in 24 months"? Nothing against ComputerLand, but I have always chaffed at this particularly dumb rule of Apple's. If you do contract programming as a consulting engineer it seems they think developership is above you. It doesn't matter if you have contracts with the US Government (the world's biggest computer buyer) Apple doesn't think you are serious.

If you aren't trying to eek out your existence selling software to "the rest of us" then they will do everything within their power to stand in your way. Never mind that the people I consult with buy the brand of computer I say the software I wrote for them runs on, period, no other factors come into the matter. I can only guess it is a hold over from the Jobs days when Apple saw itself making the next great electric toaster and thought that every home would have one. With the new and increased price of a Mac II, this camper just doesn't see the sense in it.

My comment is, no matter what stupid things Apple does, if you just stand by and they'll do something stupider soon. Thanx to good people in the Mac community (the folks on MouseHole in particular) support from Apple is just not the kind of thing you've got to have. Its easier to live without them than put up with their bulloney.

[Of all the questions on the Apple Developer Question form, the only one that they seem to pay attention to is "Will your product be on the market within the next 24 months?". If you answer no to that, you'll become a "Mailing List Class" developer, which means you'll get tech notes and newsletters, but nothing much more. - Rusty]

[I personally agree with your sentiments. Apple has always ignored the thousands of developers working in industry in vertical markets because they were not developing the next MS Word. This attitude has always bugged me. -Ed]

From: billd (Bill Dugan, Huntington Beach, CA
Subject: nVIR virus returns

We made the acquaintance of the nVIR virus recently at UC Irvine's school newspaper. We've got a 290MB file server running AppleShare, with about 7 SE's with hard drives and 4 with floppies accessing it. Shared applications. Shared viruses. Now the virus is everywhere you look. I 'vaccinated' all the

startup volumes with the May MacTutor as a reference guide. But does anybody know if nVIR is really a problem? Has anybody lost any data? (Don't quote some yokel from MacTutor or something who insists every new MacWrite release causes problems with his MacPaint 1.0.) We got the "Don't Panic" Macintalk message a couple times (after signing off NetTrek, so it happens) but that's the only effect. *[The nVIR virus does not appear to be fatal to anything but it is nasty and pervasive. Get rid of it, and I do mean all of it, at once. -Ed]*

From: rusty (Mr. Rusty Hodge, Orange, CA)

The biggest problem with nVIR is that it doesn't seem to be too compatible with 6.0.2. We were getting semi-random crashes until we realized we had it and eliminated it.

[Just when we thought that the virus epidemic was over, we were hit. Of course, this happened about 4 days before a major project deadline. Moral: always check for viruses, and back up your entire hard disk weekly - Rusty]

From: billd (Bill Dugan, Huntington Beach, CA)

All I've done so far is patch the System files, with an INIT 32 that contains a $4E75, and then deleted all the nVIR resources and replaced them with empty ones, ID 0 through 7. Haven't messed with applications yet at all. Interferon did detect nVIR on one hard drive (Virus RX did NOT!) and I found out by surprise that it eradicates the virus by deleting the applications! Whoops. *[All applications must be "cleansed" or your system will be re-infected. I said it was nasty. -Ed]*

From: adept (Roy Lovejoy, Silicon Valley)
Subject: Print Record Fields moved.

How nice of them to do this. If your print routines are not working, this may be why..

1) The number of copies set in PrJobDialog is not stored in the iCopies field of the prJob field of the print handle. (that would be too easy) It is really stored in iRowBytes field of the prXInfo field.

2) Likewise the percentage of enlargement/reduction can be found in the iBandH field of the prXInfo.

3) as per previous 'Landscape' question, you can determine portrait/landscape by looking at the rPaper field of the print handle. If rPaper.right > rPaper.left then you have Landscape mode on. (theoretically, until Apple changes that too!)

The only reason I can figure why these changes have taken place (1 & 2) is what I gleaned from a conversation with an Apple project manager in the Printer group. He basically said that they don't want developers mucking with the Print Handle themselves, only using the routines given. (if routines would be able to set all of the attributes that the PrJob & PrStyle Dialogs do, then I could understand this.. But until that day....) Hopes this helps someone.. p.s. the fields mentioned in 1 & 2 are read/write, i.e. you can change them.

From: retzes (Steven Retzlaff, La Mesa, CA)
Subject: LSC 3.0 math-library problem

I was trying to use LSC to compile a Mandelbrot-graphics program and ran into the following problem: link error: _fabs _log _sqrt . This was with 68020 and 68881 code generation enabled, _ERRORCHECK_ and _MC68881_ defined, math.h included, and the math881 library added to the project. the program usage was something like:

```
fxr = fabs(xder) ;
dist = log(x2+y*sqrt(x2+y2)/sqrt(xder*xder+yder*yder) ;
```

All the variables are locally defined as double. The program refused to compile unless I changed the function names to _log, _sqrt, and _fabs. Even when it linked, the fabs routine intermittently caused a Bus-Error-Exception and according to the LSC Debugger, failed to assign a value to the Left-Hand-Side variable fxr from fabs. The rest of the 68881 inline-operations (+,-,*,/) seem to work ok, but the math-functions don't link properly, and fabs doesn't work(I had to use a if <0 statement to emulate it).

From: robert (Rob Anthony, Chicago, IL)
Subject: C "pretty printer"

A rather non-technical question: a friend of mine is looking for a "C pretty printer, preferably with varying fonts". I don't usually program in C, suggestions?

From: thecloud (Ken Mcleod, La Habra, CA)
Subject: Re: C "pretty printer"

I have "PrettyPrint 2.0", by Andrew Shebanow... you can select font and size, as well as a number of other options (italicize comments, etc.). It allows you to pretty-print C, Pascal or assembly source.

From: chenette (Philip Chenette, LA, CA)
Subject: Opening ICONS

A question— I'm playing around with a copy of MiniEdit, the program which is developed in Stephen Chernikoff's MacIntosh Revealed. I'm trying to learn Mac programming from this. When I quit the program, I can't open any other Icons on the desktop. They select just fine, but double clicking to open just doesn't work. Rebooting solves the problem. Any thoughts on what's going on here?

From: thecloud (Ken Mcleod, La Habra, CA)

MiniEdit does something it's not supposed to do: it changes the system event mask. Look for a call to SetEventMask(theMask), where "theMask" is (I think) keyUpMask... comment out this line, and your double-click problems will go away. This was a bad blunder, considering MiniEdit is supposed to be the quintessential "example program"!

From: chenette (Philip Chenette, L A, CA)

THANKS! I tried this question on Compuserve a few weeks ago—Nary a response! *[It is not entirely MiniEdit's fault. Apple changed something in the system file a while back and several older applications started showing this problem. -Ed]*

© The Definitive MacTutor, Vol. 4

Topical Index

The Definitive MacTutor, Volume 4, 1988

Topic	Language	Title	Definitive Journal Source Page	No.	Pg.	Disk
3D	Forth	Graf3D Library Access	481	6	30	33
4th Dimension	Pascal	4th Dimension® Externals	356	9	42	36
A/UX	General	Ask Apple Technical Support	648	3	52	30
ACTA	C, Pascal	Format Drivers Bring Freedom of Revision	264	1	41	28
ADB	Pascal	Keyboard Wars!	256	1	30	28
ALAP	Forth	AppleTalk Mail	509	9	92	36
Animation	Pascal	Palette Animation	324	5	54	32
AppleTalk	Forth	AppleTalk Mail	509	9	92	36
AppleTalk Protocols	Forth	AppleTalk Mail	509	9	92	36
Assembly	HyperTalk, Assembly, Pascal	XCMD Cookbook: Assembly Interface	624	11	33	38
ATP	Forth	AppleTalk Mail	509	9	92	36
AutoHyperEdit	HyperTalk	AutoHyperEdit Moves in Groups	596	8	12	35
AutoHyperEdit	HyperTalk	ScriptTips	607	9	12	36
AutoscriptEdit	HyperTalk	HyperTools for HyperProgrammers	588	7	52	34
Background Task	C	A Background Task for Measuring Load	100	3	38	30
Benchmarks	Basic	Basic Wars and True Windows	446	10	46	37
Benchmarks	Basic	System Snooping in MS QuickBasic	449	11	17	38
Benchmarks	Fortran	The Status of Fortran	547	11	79	
Benchmarks	Lisp	MacScheme vs. Coral Lisp	552	3	72	
BitMap Rotation	Assembly	Faster BitMap Rotation	56	11	86	38
BitMap Rotation	Assembly, Pascal	Looking At Text From a Different Angle	9	2	62	29
Buttons	HyperTalk, General	The Vertical Market	627	12	71	39
Cable Pin Assignments	Pascal	Serial Port Demo	338	6	54	33
Callbacks	HyperTalk, Pascal, C	XCMD Corner	593	7	60	34
CD ROMs	HyperTalk, General	The Vertical Market	627	12	71	39
CDEF	Pascal	Pop-up Menus CDEF	364	9	56	36
Clipping	Assembly	Smoothing via PicComments	61	12	23	39
CLUT	Pascal	Palette Animation	324	5	54	32
CODE	C	A Vaccine for the 'nVIR' Virus	150	5	46	32
Code	Forth	Code Segments & Linker	525	11	22	38
Code Optimization	Assembly	Faster BitMap Rotation	56	11	86	38
Code Optimization	Assembly	Fun With Regions, Part II	380	9	82	36
CODE Resources	C	Smart Documents	93	2	86	29
Color	Basic	Color & DragGrayRgn Explored	408	1	16	28
Color	Basic	True Basic Plays the Simon Game	428	7	18	34
Color	Pascal	Graphic Devices & Chroma	303	3	58	30
Color	Pascal	MultiFinder Friendly MacDraw Plotter	274	2	16	29
Color	Pascal	Palette Animation	324	5	54	32
Color	Pascal	Screen Dump FKEY for Color Picts	295	2	52	29
Color Manager	Pascal	The Palette Manager & Color Puzzle	314	4	80	31
Color Mixer	Basic	Color Mixer Creates Custom Colors	415	4	20	31
Color Models	Basic	Hierarchical Menus & Color Notes	422	6	18	33
Color Palette	Pascal	The Palette Manager & Color Puzzle	314	4	80	31
Controls	HyperTalk, General	The Vertical Market	627	12	71	39
Data Fork	C	Smart Documents	93	2	86	29
DDP	Forth	AppleTalk Mail	509	9	92	36
Default Buttons	Basic	QuickBasic's Statement Mover	452	12	10	39
Desk Accessories	Forth	Time Manager Reminder DA	464	3	30	30
Desk Accessories	Pascal	Screen Dump FKEY for Color Picts	295	2	52	29
Desk Accessories	Pascal, Assembly	HFS Transfer DA	189	10	80	37
Dialog Filter Proc	Assembly	Font Dialog Box Using List Manager	31	7	64	34
Dialogs	Basic	QuickBasic's Statement Mover	452	12	10	39
Document Launching	C	Smart Documents	93	2	86	29
Downloader	C, PostScript	Overview & Downloader in C	79	1	78	28
DragGrayRgn	Basic	Color & DragGrayRgn Explored	408	1	16	28
Drivers	Assembly	Modifying Apple's SCSI Driver	2	1	21	28
Drivers	Assembly, C	IAC Driver, Part II	162	10	14	37
Drivers	Assembly, C	Protocols for IAC Driver, Part I	155	8	50	
Drivers	C	IAC Driver Demo in MPW C	201	11	37	38
Drivers	C, Pascal	Format Drivers Bring Freedom of Revision	264	1	41	28
Drivers	Forth	A SCSI Driver in Forth	456	2	42	29
Electronic Mail	Forth	AppleTalk Mail	509	9	92	36
Engineering	Basic	Tips for Engineering Applications	535	2	78	29
Engineering	Fortran	MatLab Comes to the Mac II	539	5	74	32
External Libraries	Forth	Graf3D Library Access	481	6	30	33

Topic	Language	Title	Definitive Journal Source Page	No.	Pg.	Disk
External Procedures	Pascal	4th Dimension® Externals	356	9	42	36
FaceIt	General	Prototyper, MacExpress, and FaceIt Compared	675	10	98	
File Format	C, Pascal	Format Drivers Bring Freedom of Revision	264	1	41	28
File Import/Export	Basic	Quick & Dirty Data Conversion	419	5	32	32
File Importing	HyperTalk, C	XCMD Corner: Importing Text	615	9	24	36
FKEY	Assembly	FKEY that runs other FKEYs!	25	6	78	33
FKEY	Pascal	Screen Dump FKEY for Color Picts	295	2	52	29
Floating Palettes	C	Tear-off Menus & Floating Palettes	112	4	26	31
Floating Point Routines	Forth	Direct 68881 Floating Point Access	469	4	72	31
Floating Windows	C	Tool Window Manager	230	12	76	39
Font Dialog	Assembly	Font Dialog Box Using List Manager	31	7	64	34
Font Map	Assembly	Font Dialog Box Using List Manager	31	7	64	34
Fonts	Assembly	Font Dialog Box Using List Manager	31	7	64	34
Games	Basic	True Basic Plays the Simon Game	428	7	18	34
Games	Pascal	The Palette Manager & Color Puzzle	314	4	80	31
Ghost Windows	C	Tool Window Manager	230	12	76	39
Graphics	Assembly	Faster BitMap Rotation	56	11	86	38
HFS	Pascal, Assembly	HFS Transfer DA	189	10	80	37
Hierarchical Menus	Basic	Color Mixer Creates Custom Colors	415	4	20	31
Hierarchical Menus	Basic	Hierarchical Menus & Color Notes	422	6	18	33
HyperCard	Basic	Quick & Dirty Data Conversion	419	5	32	32
HyperCard	HyperTalk	New HyperCard 1.2 Features	580	6	68	33
HyperCard	HyperTalk	New HyperCard Products	569	5	80	32
HyperCard	HyperTalk	PopUpMenus in HyperTalk	573	5	85	32
HyperTalk	HyperTalk, Pascal	Comparing HyperTalk to Pascal	609	9	14	36
IAC	Assembly, C	IAC Driver, Part II	162	10	14	37
IAC	Assembly, C	Protocols for IAC Driver, Part I	155	8	50	
IAC	C	IAC Driver Demo in MPW C	201	11	37	38
IAC Protocols	Assembly, C	Protocols for IAC Driver, Part I	155	8	50	
IBM	General	OS/2 Presentation Manager	681	12	46	
Icons	Pascal	Menus as Objects in TML Pascal	349	6	85	33
INIT	C	A Vaccine for the 'nVIR' Virus	150	5	46	32
INIT	Pascal	Inside Macinkeys	396	12	54	39
Input/Output	C, Pascal	Format Drivers Bring Freedom of Revision	264	1	41	28
Inter-Application-Communications	Assembly, C	IAC Driver, Part II	162	10	14	37
nter-Application-Communications	Assembly, C	Protocols for IAC Driver, Part I	155	8	50	
Inter-Application-Communications	C	IAC Driver Demo in MPW C	201	11	37	38
International Resources	Forth	Objective Activities in Forth	487	7	36	34
KCHR	Pascal	Inside Macinkeys	396	12	54	39
Key Codes	Pascal	Inside Macinkeys	396	12	54	39
Keyboard Remapping	Pascal	Inside Macinkeys	396	12	54	39
Keyboards	Pascal	Inside Macinkeys	396	12	54	39
Keyboards	Pascal	Keyboard Wars!	256	1	30	28
KMAP	Pascal	Inside Macinkeys	396	12	54	39
LaserWriter	C, PostScript	Overview & Downloader in C	79	1	78	28
Linker	Forth	Code Segments & Linker	525	11	22	38
List Manager	Forth	The Notification Manager	517	10	51	37
MacExpress	General	Prototyper, MacExpress, and FaceIt Compared	675	10	98	
MacWrite	HyperTalk, C	XCMD Corner: Importing Text	615	9	24	36
MATLAB	Fortran	MatLab Comes to the Mac II	539	5	74	
Matrices	Basic	Matrices in True Basic	442	9	32	36
Matrices	Fortran	MatLab Comes to the Mac II	539	5	74	
MC 68881	Forth	Direct 68881 Floating Point Access	469	4	72	31
MDEF	C	Tear-off Menus & Floating Palettes	112	4	26	31
MDEF	Pascal	Menus as Objects in TML Pascal	349	6	85	33
Memory	Assembly	Smoothing via PicComments	61	12	23	39
Menu Windows	Pascal	Menus In Windows	388	11	94	38
Menus	C	MenuStuff for Handling Menus	177	10	62	37
Menus	General	OS/2 Presentation Manager	681	12	46	
Menus	Pascal	Menus as Objects in TML Pascal	349	6	85	33
Menus	Pascal	Menus In Windows	388	11	94	38
Messages	General	OS/2 Presentation Manager	681	12	46	
Modeless Dialogs	Forth	The Notification Manager	517	10	51	37
Monitors	Pascal	Graphic Devices & Chroma	303	3	58	30
MPW	C, Pascal, MPW	Using Consulair C & TML in MPW	631	1	57	28
MultiFinder	Assembly	System Sleuthing II: The Sequel	669	7	26	34
MultiFinder	Assembly, C	IAC Driver, Part II	162	10	14	37
MultiFinder	Assembly, C	Protocols for IAC Driver, Part I	155	8	50	
MultiFinder	C	A Background Task for Measuring Load	100	3	38	30
MultiFinder	C	IAC Driver Demo in MPW C	201	11	37	38
MultiFinder	General	Ask Apple Technical Support	648	3	52	30

© The Definitive MacTutor, Vol. 4

Topic	Language	Title	Definitive Page	Journal No.	Pg.	Source Disk
MultiFinder	Pascal	MultiFinder Friendly MacDraw Plotter	274	2	16	29
MultiFinder	Pascal	MultiFinder Friendly MacDraw Plotter	274	2	16	29
Multiple Windows	C	Tool Window Manager	230	12	76	39
Multitasking	C	A Background Task for Measuring Load	100	3	38	30
Multitasking	Forth	Objective Activities in Forth	487	7	36	34
Navigation	VIP	Calculating Distance for Navigation	652	3	86	30
NBP	Forth	AppleTalk Mail	509	9	92	36
Notification Manager	Forth	The Notification Manager	517	10	51	37
Null Event Processing	C	A Background Task for Measuring Load	100	3	38	30
nVIR	C	A Vaccine for the 'nVIR' Virus	150	5	46	32
Object Oriented Programming	Forth	Object Forth Project	497	8	66	35
Object Oriented Programming	Forth	Objective Activities in Forth	487	7	36	34
Object Oriented Programming	Lisp	Lisp Objects: Ford vs. Chevy	565	4	92	
Objected Oriented Programming	Pascal	Menus as Objects in TML Pascal	349	6	85	33
Offset Tables	Assembly	FKEY that runs other FKEYs!	25	6	78	33
OS/2	General	OS/2 Presentation Manager	681	12	46	
PACK	Pascal, Assembly	HFS Transfer DA	189	10	80	37
PaintOne	C	Tool Window Manager	230	12	76	39
Palette Manager	Basic	True Basic Plays the Simon Game	428	7	18	34
Palette Manager	Pascal	Graphic Devices & Chroma	303	3	58	30
Palette Manager	Pascal	Palette Animation	324	5	54	32
Palettes	C	Tear-off Menus & Floating Palettes	112	4	26	31
PAP Manager	C, PostScript	Overview & Downloader in C	79	1	78	28
picComments	Assembly	Smoothing via PicComments	61	12	23	39
picLasso Comment	Pascal	Comments About PICTs	662	6	40	
PICT	Assembly	Smoothing via PicComments	61	12	23	39
PICT	Pascal	Comments About PICTs	662	6	40	
Picture Comments	Pascal	Comments About PICTs	662	6	40	
Pictures	Assembly	Smoothing via PicComments	61	12	23	39
Pictures	Basic	True Basic Plays the Simon Game	428	7	18	34
Pictures	Pascal	4th Dimension® Externals	356	9	42	36
Pictures	Pascal	Comments About PICTs	662	6	40	
Pictures	Pascal	MultiFinder Friendly MacDraw Plotter	274	2	16	29
Pictures	Pascal	Screen Dump FKEY for Color Picts	295	2	52	29
PixMap	Pascal	Comments About PICTs	662	6	40	
Plotting	Basic	Tips for Engineering Applications	535	2	78	29
Plotting	Fortran	MatLab Comes to the Mac II	539	5	74	
Polygons	Assembly	Smoothing via PicComments	61	12	23	39
Polynomials	Fortran	MatLab Comes to the Mac II	539	5	74	
PopUp Menus	Forth	Direct 68881 Floating Point Access	469	4	72	31
PopUp Menus	HyperTalk	PopUpMenus in HyperTalk	573	5	85	32
PopUp Menus	Pascal	Pop-up Menus CDEF	364	9	56	36
PostScript	C, PostScript	Overview & Downloader in C	79	1	78	28
PostScript	Pascal	Comments About PICTs	662	6	40	
Presentation Manager	General	OS/2 Presentation Manager	681	12	46	39
Printing	Assembly	Smoothing via PicComments	61	12	23	39
Printing	Fortran	Fortran Printing Interface	543	8	96	35
Printing	HyperTalk, Pascal	XCMD Cookbook: Print Manager	620	10	39	37
Printing	Pascal	MultiFinder Friendly MacDraw Plotter	274	2	16	29
Product Review	Basic, Pascal	True Basic & Prototyper Reviewed	432	8	32	35
Product Review	General	Prototyper, MacExpress, and FaceIt Compared	675	10	98	
Product Review	HyperTalk	HyperTools for HyperProgrammers	588	7	52	34
Product Reviews	HyperTalk	Happy Birthday	619	10	36	
Programming Tools	Basic, Pascal	True Basic & Prototyper Reviewed	432	8	32	35
Programming Tools	HyperTalk	New HyperCard Products	569	5	80	32
Prototyper	Basic, Pascal	True Basic & Prototyper Reviewed	432	8	32	35
Prototyper	General	Prototyper, MacExpress, and FaceIt Compared	675	10	98	
QuickBasic	Basic	QuickBasic's Statement Mover	452	12	10	39
Random Numbers	Basic	Tips for Engineering Applications	535	2	78	29
Regions	Assembly	Fun With Regions, Part II	380	9	82	36
Resource Fork	C	Smart Documents	93	2	86	29
Resource Moving	C	Smart Documents	93	2	86	29
Resources	Basic	QuickBasic's Statement Mover	452	12	10	39
Resources	Pascal	Inside Macinkeys	396	12	54	39
RGB Color	Pascal	The Palette Manager & Color Puzzle	314	4	80	31
Screen Dump	Pascal	Screen Dump FKEY for Color Picts	295	2	52	29
Screen Size	Basic	Color & DragGrayRgn Explored	408	1	16	28
Script Manager	Assembly	System Sleuthing II: The Sequel	669	7	26	34
SCSI	Assembly	Modifying Apple's SCSI Driver	2	1	21	28
SCSI	Forth	A SCSI Driver in Forth	456	2	42	29
Segmentation	Pascal, Assembly	HFS Transfer DA	189	10	80	37

Topic	Language	Title	Definitive Journal Source Page	No.	Pg.	Disk
Segments	Forth	Code Segments & Linker	525	11	22	38
Serial Port	General	MacBox and the Serial Sermon	660	5	92	
Serial Ports	Pascal	Serial Port Demo	338	6	54	33
Shell Variables	C, Pascal, MPW	Using Consulair C & TML in MPW	631	1	57	28
Show	Forth	European Developers' Conference	531	12	17	39
SIZE Resource	General	Ask Apple Technical Support	648	3	52	30
Smoothing	Assembly	Smoothing via PicComments	61	12	23	39
Solidarsoft	Forth	AppleTalk Mail	509	9	92	36
Solidarsoft	Forth	Direct 68881 Floating Point Access	469	4	72	31
Sorting	HyperTalk, Pascal	XCMD Cookbook: Sorting Routines	602	8	24	35
Sound	Pascal	4th Dimension® Externals	356	9	42	36
Statement Mover	Basic	QuickBasic's Statement Mover	452	12	10	39
Suspend/Resume	C	A Background Task for Measuring Load	100	3	38	30
Suspend/Resume	C, Pascal, MPW	Using Consulair C & TML in MPW	631	1	57	28
System	Basic	System Snooping in MS QuickBasic	449	11	17	38
System File	Assembly	System Sleuthing II: The Sequel	669	7	26	34
Task Scheduler	Forth	Objective Activities in Forth	487	7	36	34
Tear Off Menus	C	Tear-off Menus & Floating Palettes	112	4	26	31
Text Editors	C	IAC Driver Demo in MPW C	201	11	37	38
Text Styles	Forth	Formatted Text with Style	474	5	38	32
TextEdit	Assembly	Faster BitMap Rotation	56	11	86	38
TextEdit	Assembly, Pascal	Looking At Text From a Different Angle	9	2	62	29
TextEdit	C	IAC Driver Demo in MPW C	201	11	37	38
TextEdit	Forth	Formatted Text with Style	474	5	38	32
Time Manager	Forth	Time Manager Reminder DA	464	3	30	30
Tool Windows	C	Tool Window Manager	230	12	76	39
Trap Dispatcher	Assembly	QuickTrap Routines Bypass Trap Dispatcher	21	3	18	30
Traps	Assembly	QuickTrap Routines Bypass Trap Dispatcher	21	3	18	30
Traps	Assembly	System Sleuthing II: The Sequel	669	7	26	34
True Windows	Basic	Basic Wars and True Windows	446	10	46	37
Update Event	Basic	Updating Windows in Basic	412	3	24	30
Update Event	Pascal	MultiFinder Friendly MacDraw Plotter	274	2	16	29
Vaccine	C	A Vaccine for the 'nVIR' Virus	150	5	46	32
Videodiscs	HyperTalk, General	The Vertical Market	627	12	71	39
VIP	VIP	Calculating Distance for Navigation	652	3	86	30
Virus	C	A Vaccine for the 'nVIR' Virus	150	5	46	32
WaitNext Event	C	A Background Task for Measuring Load	100	3	38	30
WaitNextEvent	Pascal	MultiFinder Friendly MacDraw Plotter	274	2	16	29
WDEF	C	Tear-off Menus & Floating Palettes	112	4	26	31
windowKind	C	Tool Window Manager	230	12	76	39
Windows	Basic	Updating Windows in Basic	412	3	24	30
Windows	C	Breaking the Four Window Barrier	77	1	53	28
Windows	C	Tool Window Manager	230	12	76	39
Windows	General	OS/2 Presentation Manager	681	12	46	S
Windows	Pascal	Menus In Windows	388	11	94	38
XCMD	HyperTalk, Assembly, Pascal	XCMD Cookbook: Assembly Interface	624	11	33	38
XCMD	HyperTalk, Pascal	XCMD CookBook	584	6	73	33
XCMD	HyperTalk, Pascal	XCMD Cookbook: Sorting Routines	602	8	24	35
XCMD	HyperTalk, Pascal, C	XCMD Corner	593	7	60	34

© The Definitive MacTutor, Vol. 4

Author's Index

The Definitive MacTutor, Volume 4, 1988

Author	Language	Title	Definitive Page	Issue No.	Source Pg.	Disk
Alviani, Frank	Assembly, C	Protocols for IAC Driver, Part I	155	8	50	
Alviani, Frank	Assembly, C	IAC Driver, Part II	162	10	14	37
Alviani, Frank	C	IAC Driver Demo in MPW C	201	11	37	38
Bouldin, Chuck	Fortran	The Status of Fortran	547	11	79	
Cameron, Ray A.	Assembly	Font Dialog Box Using List Manager	31	7	64	34
Carper, Todd	Pascal	4th Dimension® Externals	356	9	42	36
Curtis, David B.	Pascal	Menus as Objects in TML Pascal	349	6	85	33
Derossi, Chris	General	Ask Apple Technical Support	648	3	52	30
Dubin, Stephen	Assembly	Fun With Regions, Part II	380	9	82	36
Dunham, David	C, Pascal	Format Drivers Bring Freedom of Revision	264	1	41	28
Fruin, Thomas	C	Tool Window Manager	230	12	76	39
Guillon, Didier	Lisp	Lisp Objects: Ford vs. Chevy	565	4	92	
Holder, John	Assembly	FKEY that runs other FKEYs!	25	6	78	33
Hopper, James	C, Pascal	Format Drivers Bring Freedom of Revision	264	1	41	28
Huxham, Fred	General	Ask Apple Technical Support	648	3	52	30
Joerding, Wayne	Forth	Object Forth Project	497	8	66	35
Kelly, Dave	Pascal	MultiFinder Friendly MacDraw Plotter	274	2	16	29
Kelly, Dave	Basic	Color & DragGrayRgn Explored	408	1	16	28
Kelly, Dave	Basic	Updating Windows in Basic	412	3	24	30
Kelly, Dave	Basic	Color Mixer Creates Custom Colors	415	4	20	31
Kelly, Dave	Basic	Quick & Dirty Data Conversion	419	5	32	32
Kelly, Dave	Basic	Hierarchical Menus & Color Notes	422	6	18	33
Kelly, Dave	Basic	True Basic Playes the Simon Game	428	7	18	34
Kelly, Dave	Basic, Pascal	True Basic & Prototyper Reviewed	432	8	32	35
Kelly, Dave	Basic	Matrices in True Basic	442	9	32	36
Kelly, Dave	Basic	Basic Wars and True Windows	446	10	46	37
Kelly, Dave	Basic	System Snooping in MS QuickBasic	449	11	17	38
Kelly, Dave	Basic	QuickBasic's Statement Mover	452	12	10	39
Korn, Peter	C	A Background Task for Measuring Load	100	3	38	30
Koscheka, Donald	HyperTalk, Pascal	XCMD CookBook	584	6	73	33
Koscheka, Donald	HyperTalk, Pascal, C	XCMD Corner	593	7	60	34
Koscheka, Donald	HyperTalk, Pascal	XCMD Cookbook: Sorting Routines	602	8	24	35
Koscheka, Donald	HyperTalk, C	XCMD Corner: Importing Text	615	9	24	36
Koscheka, Donald	HyperTalk, Pascal	XCMD Cookbook: Print Manager	620	10	39	37
Koscheka, Donald	HyperTalk, Assembly, Pascal	XCMD Cookbook: Assembly Interface	624	11	33	38
Langowski, Jörg	Assembly	Modifying Apple's SCSI Driver	2	1	21	28
Langowski, Jörg	Forth	A SCSI Driver in Forth	456	2	42	29
Langowski, Jörg	Forth	Time Manager Reminder DA	464	3	30	30
Langowski, Jörg	Forth	Direct 68881 Floating Point Access	469	4	72	31
Langowski, Jörg	Forth	Formatted Text with Style	474	5	38	32
Langowski, Jörg	Forth	Graf3D Library Access	481	6	30	33
Langowski, Jörg	Forth	Objective Activities in Forth	487	7	36	34
Langowski, Jörg	Forth	Object Forth Project	497	8	66	35
Langowski, Jörg	Forth	AppleTalk Mail	509	9	92	36
Langowski, Jörg	Forth	The Notification Manager	517	10	51	37
Langowski, Jörg	Forth	Code Segments & Linker	525	11	22	38
Langowski, Jörg	Forth	European Developers' Conference	531	12	17	39
Lieske, Jay	Fortran	Fortran Printing Interface	543	8	96	35
Luckie, Bill	VIP	Calculating Distance for Navigation	652	3	86	30

© The Definitive MacTutor, Vol. 4

Author	Language	Title	Definitive Page	Issue No.	Pg.	Source Disk
Marriott, Greg	Pascal	Graphic Devices & Chroma	303	3	58	30
Matthews, James	Pascal	Menus In Windows	388	11	94	38
McNamara, Joel	C	Smart Documents	93	2	86	29
Melton, Don	C	Tear-off Menus & Floating Palettes	112	4	26	31
Merrill, Paul F.	HyperTalk, Pascal	Comparing HyperTalk to Pascal	609	9	14	36
Moore, Thomas W.	Assembly	Fun With Regions, Part II	380	9	82	36
Morton, Mike	Assembly	QuickTrap Routines Bypass Trap Dispatcher	21	3	18	30
Morton, Mike	Assembly	Faster BitMap Rotation	56	11	86	38
Olsen, John D.	Assembly, Pascal	Looking At Text From a Different Angle	9	2	62	29
Pavkovic, Nicholas	C, PostScript	Overview & Downloader in C	79	1	78	28
Plamondon, James	Pascal	Pop-up Menus CDEF	364	9	56	36
Poland, Doug	General	Prototyper, MacExpress, and FaceIt Compared	675	10	98	
Powell, William	C, Pascal, MPW	Using Consulair C & TML in MPW	631	1	57	28
Rausch, William	C	Breaking the Four Window Barrier	77	1	53	28
Ritter, Mike	C	Tear-off Menus & Floating Palettes	112	4	26	31
Samuel, Fabien	Pascal	Inside Macinkeys	396	12	54	39
Scanlin, Mike	C	A Vaccine for the 'nVIR' Virus	150	5	46	32
Scheiderich, Tom	Pascal	Serial Port Demo	338	6	54	33
Sheets, Steven C.	Pascal	Keyboard Wars!	256	1	30	28
Sheets, Steven C.	Pascal	Screen Dump FKEY for Color Picts	295	2	52	29
Sheets, Steven C.	Pascal	The Palette Manager & Color Puzzle	314	4	80	31
Sheets, Steven C.	Pascal	Palette Animation	324	5	54	32
Shirley, Mark	C	MenuStuff for Handling Menus	177	10	62	37
Smith, David E.	Pascal	MultiFinder Friendly MacDraw Plotter	274	2	16	29
Snively, Paul	Assembly, C	Protocols for IAC Driver, Part I	155	8	50	
Snively, Paul	Assembly, C	IAC Driver, Part II	162	10	14	37
Snively, Paul	Lisp	MacScheme vs. Coral Lisp	552	3	72	
Stauder, Fred	HyperTalk	New HyperCard Products	569	5	80	32
Stauder, Fred	HyperTalk	PopUpMenus in HyperTalk	573	5	85	32
Stauder, Fred	HyperTalk	New HyperCard 1.2 Features	580	6	68	33
Stauder, Fred	HyperTalk	HyperTools for HyperProgrammers	588	7	52	34
Stauder, Fred	HyperTalk	AutoHyperEdit Moves in Groups	596	8	12	35
Stauder, Fred	HyperTalk	ScriptTips	607	9	12	36
Stauder, Fred	HyperTalk	Happy Birthday	619	10	36	
Stauder, Fred	HyperTalk, General	The Vertical Market	627	12	71	39
Story, Clifford	Pascal, Assembly	HFS Transfer DA	189	10	80	37
Tabasso, Raul	Assembly	Smoothing via PicComments	61	12	23	39
West, Joel	Pascal	Comments About PICTs	662	6	40	
West, Joel	Assembly	System Sleuthing II: The Sequel	669	7	26	34
Weston, Dan	General	OS/2 Presentation Manager	681	12	46	
Willis, Reverend	General	MacBox and the Serial Sermon	660	5	92	
Zarchan, Paul	Basic	Tips for Engineering Applications	535	2	78	29
Zarchan, Paul	Fortran	MatLab Comes to the Mac II	539	5	74	32

© The Definitive MacTutor, Vol. 4

$29.95 U.S. / $37.50 Canada

The Definitive

MacTutor

The Macintosh Programming Journal

VOLUME 4

52995

780961 754433